VASILE SURDUCAN WOUTER VAN OOIJEN

MICROCONTROLERE PENTRU TOȚI



1001111010010110

ediția a 2-a
completată și revizuită

Vasile Surducan **Wouter van Ooijen**

MICROCONTROLERE PENTRU TOȚI

ediția a 2-a
completată şi revizuită

Editura Mâna Autorului
ediția electronică 2006

Familiei mele,
cu cel mai mare drag,
pentru că mă suportă şi mă iubesc
aşa cum sunt…

Contribuția autorilor: Vasile Surducan cap.1,2,3,4,5,6,7
Wouter van Ooijen cap.1,2,3

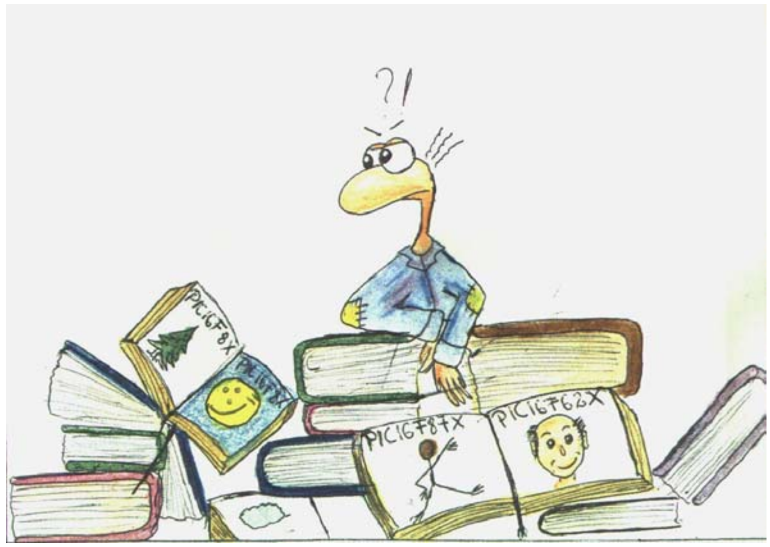
Caricaturi: Mădălina Surducan cap.1,3,4,5,6,7
Ileana Surducan cap.2

Desen copertă: Maria Surducan

CUPRINS:

1 Pornim aventura

1.1 Ce trebuie să ştim ?



Novicele care se intersectează pentru prima dată cu noțiunea de microcontroler este tentat în exuberanța sa, să finalizeze cu nerăbdare o aplicație pe care o consideră interesantă şi simplă la prima vedere, însă constată pe parcursul realizării ei că obstacolele neprevăzute întâlnite sunt dificile. Depăşirea acestora cu brio implică eforturi deosebite în însuşirea cunoştințelor de electronică generală, (pentru specialistul în **software**) respectiv a celor de algoritmi numerici, conversii în şi din diverse baze de numerație, operații matematice, etc. (pentru specialistul în **hardware**). Ideal este ca cel ce se aventurează pe tărîmul "combinatei" hardware-software cu microcontrolere să posede cunoştințe detaliate în ambele domenii. Nu disperați dacă vi se pare că aparțineți cu precădere domeniului software. Nu plângeți nici dacă va simțiti mai mult hard-ist. Singurul lucru care nu trebuie să vă lipsească este curajul, restul vine de la sine pe parcursul parcurgerii acestei cărți şi a documentației anexate pe CD sau WEB. Nu am pretenția că informația din această carte este suficientă pentru a deveni expert în microcontrolere. Cititorul poate însă încerca să afle pe propria sa piele…

1.2 Ce este un microcontroler flash şi ce mai avem nevoie…

Privit din exterior, microcontrolerul *mid-range* produs de Microchip (de care ne vom ocupa în acestă carte) este un circuit integrat ordinar cu 8 până la 68 de pini având diferite tipuri ale capsulei. Din punct de vedere al apartenenței la domeniul electronicii analogice sau digitale, este un hibrid conținând atât elemente analogice (eşantionare-memorare, convertoare analogic-digitale, comparatoare, referințe de tensiune) cât şi elemente digitale complexe specifice microprocesoarelor şi sistemelor de dezvoltare (memorie RAM-volatilă, memorie EEPROM-nevolatilă, temporizatoare, regiştrii cu funcții variate: **P**uls **W**ith **M**odulation – modulație cu lărgime de puls, **U**niversal **S**ynchronous **A**synchronous **R**eceiver **T**ransmiter – transmițător/receptor universal sicron/asincron etc.). Ceea ce deosebeşte esențial un microcontroler de un circuit integrat analogic sau digital este faptul că el nu valorează aproape nimic atât timp cât nu este programat, mai mult, neprogramat nu funcționează nici măcar oscilatorul acestuia! Programul software îi conferă aceluiaşi sistem cu microcontroler, puterea de a avea utilități diferite deşi schema hardware rămîne aproape neschimbată.

Avantajul unui microcontroler *flash* față de unul clasic ***O**ne **T**ime **P**rogramable* sau cu ştergere prin expunere la radiație ultravioletă, este posibilitatea de a rescrie memoria

program a acestuia de cel puțin 10000 de ori. Dacă programul nu funcționează din prima încercare (e miracol dacă funcționează!) avem posibilitatea modificării acestuia şi rescrierea lui în memoria program a microcontrolerului. Inevitabil, editarea şi testarea unui program, necesită cunoştințe medii de programare şi existența unor dispozitive ajutătoare numite "unelte de dezvoltare". Acestea sunt: **programatorul, editorul, compilatorul, simulatorul, bootloaderul,** şi eventual **emulatorul**.

- Programatorul (compus din hardware+software) transferă fila hexagesimală rezultată în urma compilării filei sursă (programul scris de utilizator), în memoria program a microcontrolerului, de asemenea poate programa memoria EEPROM şi "fuzibilele" de configurare ale microcontrolerului. Fuzibilele sunt conținute în registrul *configuration word* şi conțin informații variate privind oscilatorul, protecția memoriei, *reset*-ul, etc.
- Editorul permite scrierea codului sursă a programului utilizator. Este un program software evoluat ce rulează pe PC şi uşurează scrierea programului jal sau assembler.
- Compilatorul transformă codul sursă în cod hexagesimal standardizat, recunoscut de microcontroler.
- Simulatorul este un program software în care se importă fila hexagesimală şi/sau codul sursă şi care permite verificarea pas cu pas a corectitudinii acestuia, inspectând regiştrii ce "mimează" funcționarea microcontrolerului.
- Bootloaderul (compus din hardware+software) transferă rapid codul hexazecimal în microcontroler, utilizând un program rezident de boot ce rulează în microcontroler şi un program software în PC. Este util doar la faza de prototip a unui produs cu microcontroler, deci inevitabil şi în procesul de învățare al utilizării microcontrolerului.
- Emulatorul (compus din hardware+software) este un înlocuitor fizic pentru microcontroler şi se află sub directa coordonare în timp real a PC-ului, conectorul emulatorului se introduce direct în soclul microcontrolerului utilizat de aplicația noastră, înlocuindu-l la faza de testare a programului.

Fără a avea acces la instrumentele de dezvoltare ce implică atât hardware cât şi software (programator, emulator, bootloader) care de obicei sunt scumpe (de la 20$ la 3000$), începătorul într-ale microcontrolelor poate să se descurce foarte bine construindu-şi singur minimul de accesorii necesare şi să finalizeze într-un timp record aplicația dorită. Ajutorul neprețuit pe care acesta îl are este rețeaua WEB.

1.2.1 Aspectul microcontrolerului

Dacă aveți în acest moment un sentiment tulbure de neîmplinire, înseamnă că sunteți într-adevăr începător şi trebuie să vă familiarizați şi cu modul de împachetare al cipului microcontroler într-o capsulă uşor de utilizat. Pentru că singura capsulă ce permite inserarea şi extragerea în/din soclu este capsula DIP (Dual Inline Pin -adică două linii de terminale) numai aceasta va fi imortalizată în imaginile următoare pentru câteva tipuri de microcontrolere *flash mid range*. Prefixul P (PDIP) evidențiază că este vorba de o capsulă DIP din plastic. Această capsulă poate fi şi din ceramică, sau pentru microcontrolerele cu ştergere prin radiație UV(ultra-violete) poate avea o fereastră de cuarț optic. Capsula DIP nu este cea mai grozavă, deoarece ocupă un spațiu mare pe cablajul imprimat (PCB), însă este cea care se utilizează la faza de realizare a unui prototip sau unicat. Pentru un produs de serie, mai utilizate sunt capsulele SOIC sau TQFP care sunt mult mai mici.

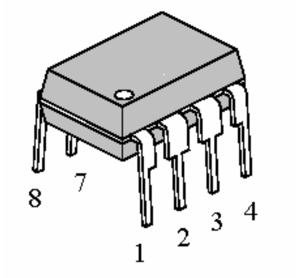


Fig.1-1 Aspectul capsulei DIP8

Numerotarea pinilor începe întotdeauna din stânga jos şi continuă în sens antiorar pe linia următoare de pini. Pinul 1 este marcat fie cu un punct (ca în imagine), fie cu o degajare semicirculară la mijlocul distanţei dintre pinii 1 şi 8. Marcajul este realizat din matriţă, la turnarea răşinii în capsulă. Pinii de programare ICSP-HVP sunt: ICSPDAT-7; ICSPCLK-6; VPP-4; VDD-1; VSS-4.

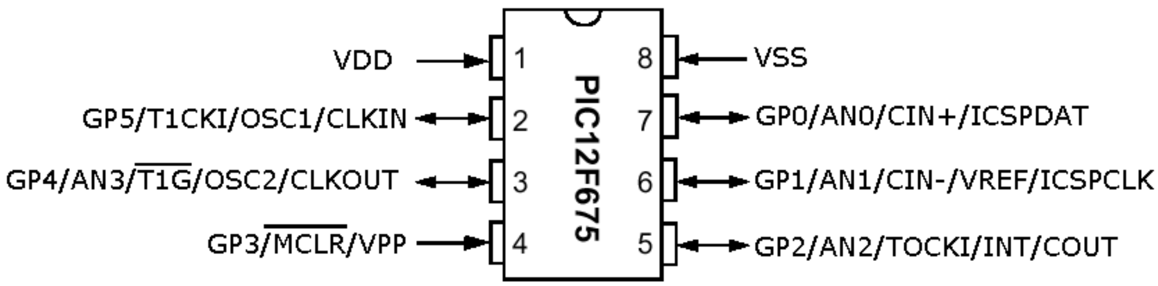


Fig. 1-2 Descrierea pinilor microcontrolerului PIC12F675

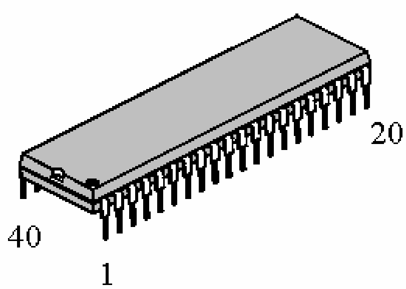


Fig.1-3 Aspectul capsulei DIP40 cu lăţimea de 600mil

Capsula PDIP40 (Plastic DIP) pentru PIC16F877/874 are lăţime dublă (600mil [mili-inch] = approximativ 15mm) comparativ cu capsula de 300 mil (7.5mm)

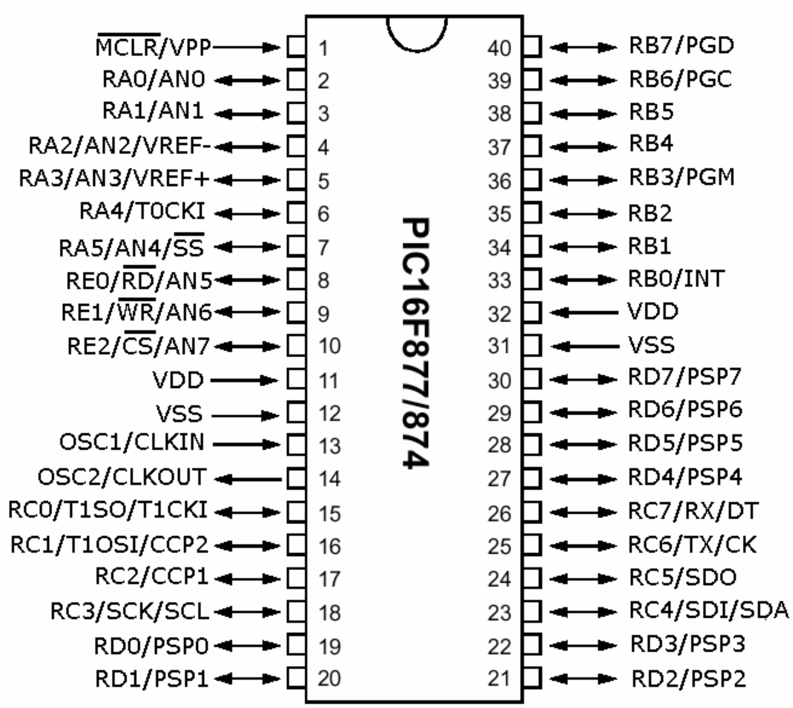


Fig.1-4 Funcţiile pinilor microcontrolerului PIC16F877/874, capsula PDIP40**.** Pinii microcontrolerului au funcţii multiple în mod secvenţial, ei nu pot îndeplini toate funcţiile prezentate în acelaşi timp!
De exemplu RC6 are rolul intrarea sincronă de tact CK sau transmisie de date TX în comunicaţia serială sau pin de uz general de intrare ieşire.
Pinii de programare ICSP-HVP sunt: PGD-40; PGC-39; VPP-1; VDD-11, 32; VSS-12,31.

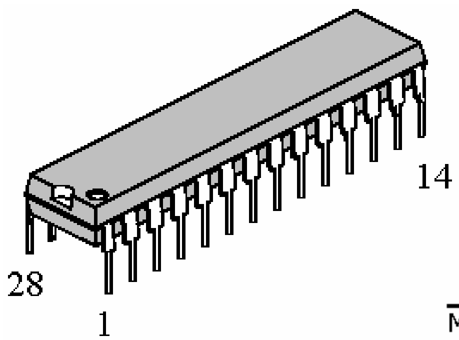
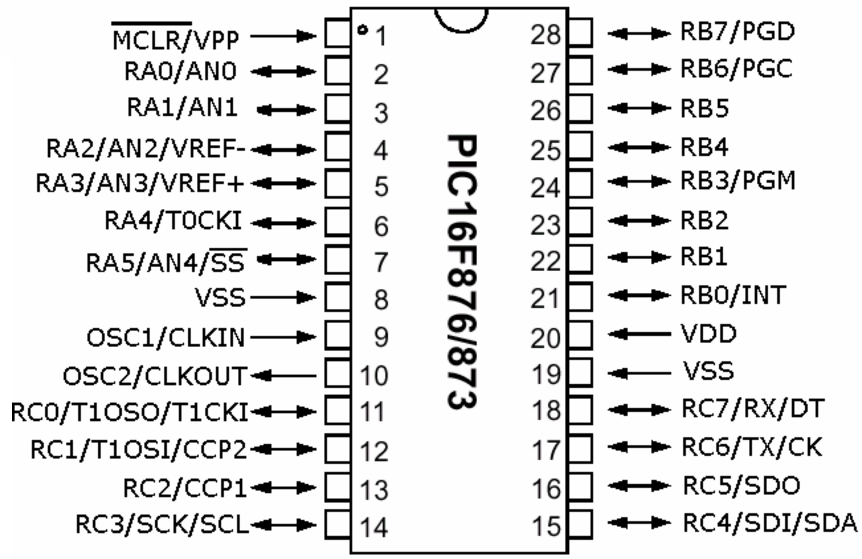


Fig.1-5 Aspectul capsulei DIP 28 de 300 mili-inch

Capsulele DIP14, DIP16, DIP18 sunt similare cu aceasta, singura diferență fiind numărul de pini şi implicit lungimea capsulei.

Fig.1-6 Semnificația pinilor microcontrolerului PIC16F873/876, PDIP28, pinii de programare ICSP-HVP sunt:
PGD-28 (data);
PGC-27(clock);
VPP-1(+13.5V);
VDD-20 (+5V);
VSS-8,19 (GND)



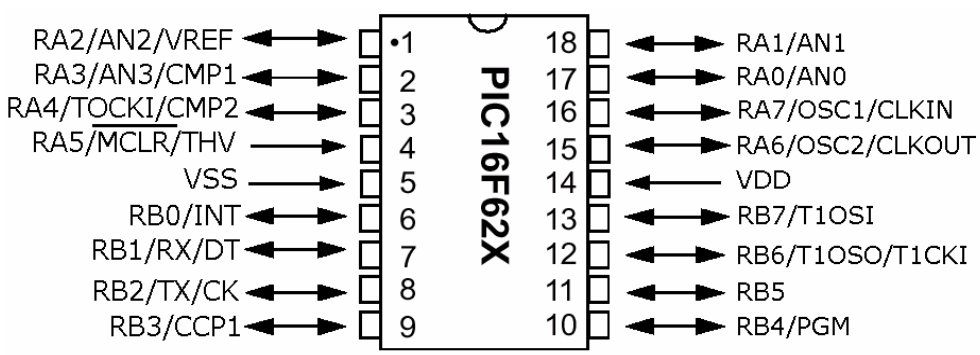


Fig.1-7 Semnificația pinilor PIC16F627/628, capsula PDIP18, pinii de programare ICSP-HVP: ICSPDAT-13; ICSPCLK-12; VPP-4; VDD-14; VSS-5.

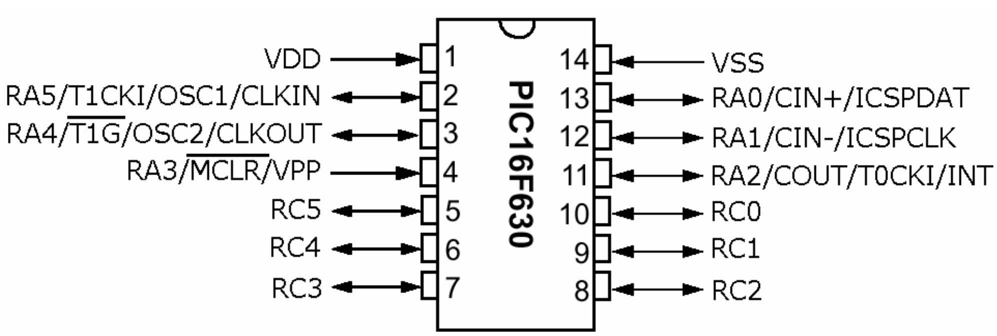


Fig.1-8 Semnificația pinilor PIC16F630/PIC16F676, capsula PDIP14

Impresia de nebuloasă pe care o dă utilizatorului semnificația multiplă a funcției fiecărui pin poate fi uşor corectată dacă se are în vedere secvențialitatea acestei funcții. De exemplu, studiind o clipă [fig.1-8], se observă trei clase de pini pentru PIC16F630/676:

- Pini cu o singură funcție:
  VDD - tensiunea de alimentare pozitivă +2…+5V (vezi fila de catalog pe CD!)
  VSS - masa tensiunii de alimentare
  RC4, RC5 - pini de uz general cu funcție digitală de intrare-ieşire
- Pini cu cel puțin două funcții (numai pentru PIC16F676):
  RC3/AN7;RC2/AN6;RC3/AN5;RC0/AN4 – pini cu funcție dublă de intrare-ieşire de uz general **sau** de intrare analogică pentru convertorul Analogic Digital intern
- Pini cu mai mult de două funcții (convertorul AD e disponibil numai în PIC16F676):
  RA0/AN0/CIN+/ICSPDAT – pin cu funcție de intrare-ieşire **sau** intrare analogică **sau** intrare neinversoare de comparator **sau** intrare de date pentru programare ICSP
  RA1/AN1/CIN-/VREF/ICSPCLK – pin cu funcție de intrare-ieşire **sau** intrare analogică **sau** intrare inversoare de comparator **sau** intrare pentru tensiune de referință pentru convertorul AD **sau** intrare de tact pentru programare ICSP.
  RA2/AN2/COUT/T0CKI/INT – pin cu funcție de intrare-ieşire **sau** intrare analogică **sau** ieşire de comparator **sau** intrare de tact extern pentru TMR0 **sau** intrare de intreruperi externe.
  RA5/T1CKI/OSC1/CLKIN – pin cu funcție generală de intrare-ieşire **sau** intrare de tact pentru temporizatorul TMR1 **sau** pin de intrare pentru oscilatorul extern cu cuarț, (sau rezonator sau **R**ezistență şi **C**ondensator) **sau** pin de intrare pentru un semnal de tact de la un oscilator extern.
  RA4/T1G/OSC2/AN3/CLKOUT – pin cu funcție de intrare-ieşire **sau** intrare de validare a tactului pentru TMR1 **sau** intrare analogică **sau** ieşire de oscilator pentru oscilator extern cu cuarț.
  RA3/MCLR/VPP – intrare de uz general **sau** intrare de reset **sau** tensiune de programare pentru HVP.

Terminologia necunoscută va fi explicată parțial în cursul capitolelor următoare. Cu toate acestea, se consideră că cititorul deține cunoştințe minime de electronică şi că următorii termeni au mai fost întâlniți:

- Tensiune de referință: sursă de tensiune cu impedanță de ieşire mininimă, a cărei stabilitate nu este afectată de temperatura ambiantă şi de zgomot.
- Comparator: amplificator operațional fără reacție negativă (sau cu reacție pozitivă).
- Convertor AD: sistem electronic ce transformă o mărime de intrare analogică într-o mărime de ieşire digitală.

1.2.2 Construcția programatorului

Programatorul este dispozitivul indispensabil orcărui specialist în **“embedded software”** adică software dedicat aplicației ce conține un hardware inteligent. Programatorul se compune dintr-un modul electronic ce realizează interfațarea între calculatorul PC şi aplicația conținând microcontrolerul, şi un program software ce rulează pe PC într-un sistem de operare preferat de utilizator. După modul de conectare la calculator pot fi definite trei tipuri de programatoare, ce poartă denumirea celor care le-au imaginat

pentru prima dată: programatorul paralel (sau de tip **David Tait**) tratat şi în nota de aplicaţie elaborată de Microchip AN-589 [1], programatorul serial (există divergenţe de opinii privind numele primului "inventator" al acestui tip de programator, este cunoscut pe internet ca JDM, NOPPP, PONY) şi programatorul USB [20]. Programatoarele pot fi destinate pentru prototipuri sau pentru producţia de serie.

Rolul programatorului este acela de a transfera în memoria program a microcontrolerului, fila hexa ce conţine munca dvs. în format compilat. Deoarece este necesară multă experienţă pentru a crea un program funcţional "în doi timpi şi trei mişcări", experienţă care se dobândeşte în ani de muncă, numărul înscrierilor succesive într-un microcontroler poate atinge zeci sau chiar sute de ori până la obţinerea efectului scontat. Este evident că visul orcărui utilizator de microcontrolere este utilizarea unui programator simplu de utilizat şi care să solicite numărul minim de manevre.

1.2.3 Programatorul paralel în regim prototip

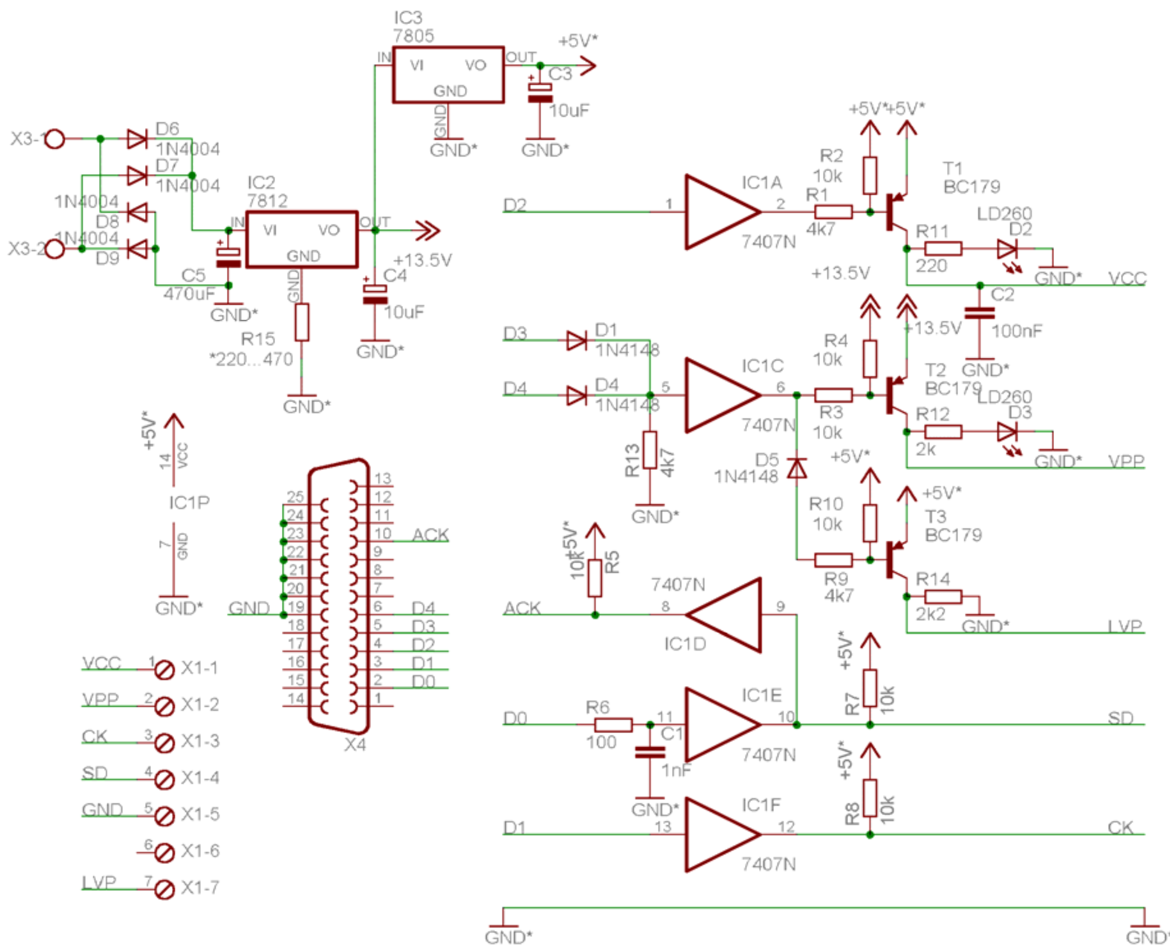


Fig. 1-9 Programator paralel, schema de principiu

Programatorul paralel utilizează interfaţa paralelă a PC-ului (**LPT**) pentru transferul datelor. Ce este şi care sunt modurile de funcţionare ale interfeţei paralele a calculatorului personal, puteţi afla navigând pe CD:\PC_info\ sau [2]. După cum se poate constata, programatorul

paralel pare la prima vedere destul de “stufos”, însă o inspecție atentă a schemei va evidenția simplitatea funcționării acesteia [fig.1-9].

Semnalele utilizate de programator (D0…D4 şi ACK) care sunt preluate din interfața paralelă LPT, sunt separate de aceasta cu bufere open colector TTL de tip 407 sau 417 (tensiunea V_{CE} a tranzistorului open colector trebuie să fie de minim 15V). D3 şi D4 sunt conectate ca SAU logic pentru a putea utiliza corect diverse programe software (de exemplu pachetul software **propic2** [3] utilizează pentru programarea microcontrolerului PIC16F84/PIC16F628 bitul D4 iar pentru PIC16F877 bitul D3 al interfeței paralele). D2 şi D3 pot fi înlocuite cu un ştrap corespunzător între pinul 5-IC1A şi pinii D3 sau D4 ai conectorului X1. Rețeaua R6, C1 este extrem de importantă dacă se lucrează cu calculatoare rapide în care zgomotul datorat vitezelor de comutare ridicate se propagă pe ieşirile de date. Această rețea de întârziere asigură un front pe semnalul de date fără supracreşteri importante. Lipsa acestuia va face ca o parte din software-urile de programare să genereze eroare de scriere la adresa zero (prima adresă verificată) mai ales dacă software-ul utilizează algoritmul rapid de înscriere (utilizat pentru PIC16FxxxA). Corectitudinea semnalului de date este verificată prin întoarcerea acestuia pe **ack**nowledge (ACK). Semnalele de programare ajung la microcontroler prin conectorul X2 numit ICSP (**I**n **C**ircuit **S**erial **P**rogramming). Microcontrolerele PIC acceptă o modalitate modernă de programare direct în circuitul pe care îl vor deservi, soclul pentru capsula PDIP este opțional, el nu este necesar dacă fiecare aplicație va avea conectorul ICSP montat. O proiectare inteligentă va permite ca aplicației să i se poată modifica firmware-ul, fără a extrage microcontrolerul din soclu şi al muta pe soclul unui programator. Repetarea acestei manevre de un număr nedefinit de ori (situație inevitabilă pentru un începător) duce la ruperea pinilor terminali (de exemplu aceştia pot fi 1,9 şi 10,18 la un microcontroler cu capsula PDIP de 18 pini). De aceea conectorul X2 este perechea tată a conectorului ICSP care va echipa fiecare placă de test pe care o veți construi.

Semnalele de tact [fig.1-9] (**CK, c**loc**k**) şi de date (**SD, s**erial **d**ata) sunt aplicate pe pinii corespunzători ai microcontrolerului: RB6 (**clock**) respectiv RB7 (**data**). Tensiunile de alimentare VCC=5V, respectiv de programare VPP=13…14V, se aplică la momentul programării, pe pinii de alimentare VDD respectiv pe **MCLR** (**reset**). Pentru generarea acestora se folosesc două tranzistoare pnp, T1 şi T2. Blocarea acestora când tensiunea de comandă are nivel logic *high,* este asigurată de R2 şi R4. Când semnalul de comandă D2 respectiv D3 sau D4 este *low***,** tranzistoarele intră în conducție asigurând tensiunile de alimentare şi programare în secvența dată de specificația tehnică de programare pe care o puteți studia din: [4] CD:\datasheet\microchip\pic16c84prog.pdf sau
[5] CD:\datasheet\microchip\pic16f8xxprog.pdf.

Pentru modul de programare cu tensiune redusă (**LVP, l**ow **v**oltage **p**rogramming) se utilizează o schemă identică cu cea descrisă anterior, formată din T3 şi rezistențele R9, R10, R14. Comanda este asigurată prin D1 (necesară pentru a separa comenzile unor potențiale diferite), din acelaşi punct în care se comandă obținerea **HVP** (**h**igh **v**oltage **p**rogramming), se observă că programatorul va genera tot timpul atât **HVP** cât şi **LVP**, utilizatorul fiind acela care va alege tipul de tensiune de programare după cum aplicația lui o va cere. Conectorul ICSP este de fapt jumătate dintr-un soclu pentru circuit integrat de 14 pini DIP, se recomandă să fie de bună calitate (AUGAT, aurit) pentru a permite un număr mare de conectări fără pene de contact. Pinul x1-6 va deveni cheia acestui conector prin tăierea lui pe conectorul tată (înspre programator) şi umplerea sa cu fludor pe conectorul mamă (conectorul dinspre microcontroler). Acest lucru va duce la imposibilitatea inversării conectorului ICSP la programare şi implicit a deteriorării microcontrolerului. Conectorul

tată poate fi unul special sau chiar cealaltă jumătate din conectorul Augat 2x14 pini, folosit în sens invers; pinii care în mod normal se cositoresc în placa PCB devin conectori pentru inserţie iar locaşurile pentru pinii circuitului integrat devin punctele de cositorire a panglicii. Acest tip de conexiune este extrem de ieftină (e nevoie de două barete Augat de 7 pini fiecare) şi foarte robustă; mult mai bună decât utilizând perechi specializate mamă-tată de conectori în linie de 2.54mm. Notaţi că doar conectorii AUGAT permit acest tip de conexiune! (secţiunea pinilor acestora este circulară cu diametrul de cca 0.9mm). Semnalizarea optică a programării se face cu led-ul D3 (roşu) pentru tensiunea de programare, respectiv D2 (verde) pentru tensiunea de alimentare.

Există scheme echivalente care nu utilizează tranzistori pnp pentru generarea potenţialelor de programare ci comutatoare CMOS de tipul MMC4016/4066 sau 4051. De asemenea există programatoare care utilizează inversoare de tip SN7406 în locul repetoarelor SN7407. Funcţionarea acestora este similară cu schema prezentată dar necesită inversarea comenzilor. Multe produse software destinate comenzii programatoarelor acceptă setarea parametrilor în funcţie de programatorul construit [6] sau chiar schimbarea prin program a biţilor de comandă [7].

Tensiunile de alimentare de +5V, respectiv tensiunea HVP de +13…14V sunt obţinute simplu dintr-o tensiune redresată şi filtrată, ţinând cond de condiţia minimă de bună funcţionare pe care o cer stabilizatoarele monolitice cu trei terminale din seria 78xx şi anume, menţinerea unei diferenţe minime între tensiunea de intrare şi cea stabilizată de ieşire de cel puţin 2…4V. Deoarece nu există stabilizatoare cu tensiune standardizată de ieşire de 13,5V se utilizează un artificiu prin flotarea stabilizatorului IC3 la cca. 1,5V prin montarea unei rezistenţe pe pinul **Iadj** (**adj**ustment current), pin care în mod normal se conectează la masă. Această rezistenţă se determină experimental deoarece fiecare lot de stabilizatoare are o marjă de eroare importantă a acestui parametru. Valoarea rezistenţei R15 este cuprinsă între 220 şi 470 de ohmi. De remarcat faptul că stabilizatorul IC3 disipă atât puterea consumată pe pinul de programare (+13,5V), cât şi cea consumată pe pinul de alimentare al microcontrolerului (+5V), motiv pentru care poate necesita un mic radiator.

Acest tip de programator funcţionează bine chiar dacă se utilizează facilitatea ICSP fără a asigura tensiunea de alimentare montajului în care microcontrolerul este montat! Acest lucru înseamnă că întregul curent de alimentare al montajului realizat (pentru tensiunea de +5V) va fi asigurat numai de programator. Limita rezonabilă pentru programarea corectă în situaţia descrisă, este un consum maxim de 100mA din programator pentru tensiunea de +5V (dimensionarea transformatorului de alimentare se va face în mod corespunzător).

Hardware-ul prezentat este gândit pentru a funcţiona cu o multitudine de programe software ce pot fi obţinute în regim **freeware** de pe internet (adresele sunt valabile pentru momentul editării cărţii) sau CD:\programatoare\, care sunt total sau parţial compatibile cu el: http://www.melabs.com http://www.propic2.com http://www.ic-prog.com http://www.lpilsley.co.uk

Concluzii: *Programatorul paralel este un programator cu gabarit relativ mare, având un transformator de reţea încorporat, asigură un curent de programare mare şi o bună izolare între aplicaţie şi calculator fiind controlat de o gamă largă de produse software obtenabile gratuit de pe internet. Acceptă programarea tuturor microcontroleror Microchip şi a unor memorii eeprom seriale.*

1.2.4 Programatorul serial în regim prototip

Programatorul serial utilizează facilitatea de a "fura" energia necesară funcționării direct din interfața serială a calculatorului. Acest mod, care pare periculos la prima vedere nu produce defecțiuni calculatorului dacă este utilizată cu precauție. Semnalele interfeței seriale au autolimitare în majoritatea arhitecturilor de PC, însă un utilizator inteligent nu se va baza niciodată pe această afirmație. Numai laptop-urile utilizează circuite de interfațare RS-232 de tipul charge-pump (tensiunile necesare interfațării sunt generate de un oscilator local, vezi de exemplu fila de catalog a circuitului MAX232, [8] CD:\datasheet\maxim\max232.pdf). O concluzie firească conduce la ideea că pe unele laptop-uri este posibil ca programatorul serial să nu funcționeze de loc, deoarece curentul debitat de acestea este insuficient pentru a alimenta programatorul şi a genera curentul de programare.

Semnalele utilizate pentru programator sunt: TxD generează Vpp, DTR generează SD (**S**erial **D**ata); verificarea semnalului de date se face prin întoarcere pe CTS, RTS genereaza CK (**C**loc**K**). Pentru a înțelege în detaliu funcționarea acestui programator este necesar să ne familiarizăm cu nivelele logice cât şi cu denumirile specifice ale semnalelor interfeței seriale [9] (CD:\PC_info\com.pdf) aruncând o privire şi în capitolul 6. Pentru un receptor de semnal RS232, un nivel 1 logic înseamnă o tensiune de ieşire cuprinsă între –3V şi -15V, respectiv un nivel logic 0, o variație a tensiunii între +3V şi +15V. Nivelele maxime la emisie pentru emițătorul RS232 se găsesc în cazul laptop-urilor la limita de ±8V…±10V. De reținut că nivelele logice cu care funcționează microcontrolerul sunt compatibile TTL pentru o tensiune de alimentare de +5V a acestuia, respectiv se poate considera teoretic 0 logic, un nivel de 0V iar 1 logic un nivel de 5V (realitatea este după cum ştiți diferită, existând un domeniu de tensiuni pentru ambele nivele logice). Pentru a evidenția faptul că semnalele interfeței seriale nu sunt compatibile cu microcontrolerul am menționat în mod exasperant, explicit, care sunt nivelele de tensiune implicate.

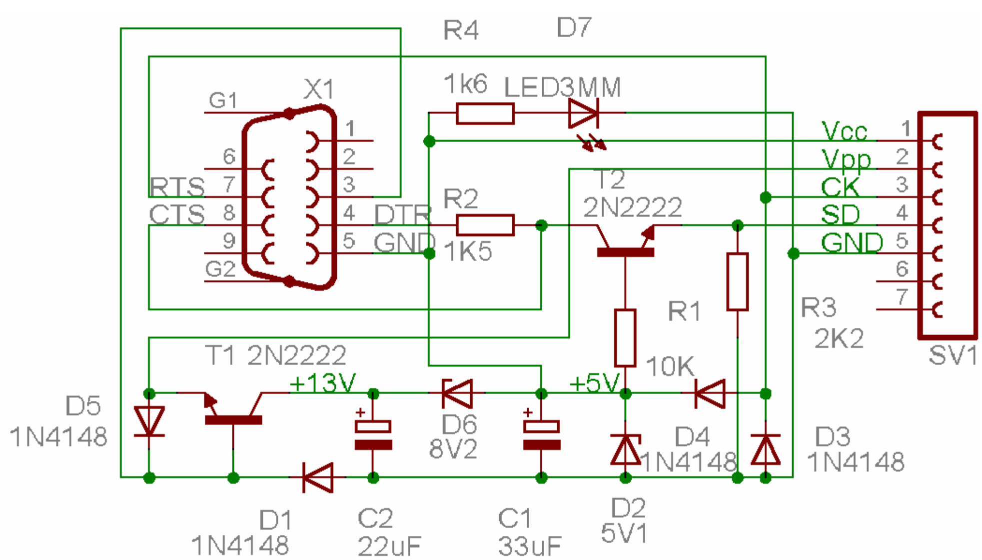


Fig.1-10 Programator serial, schema de principiu

Exista o varietate de clone ce utilizează principiul prezentat anterior. O idee ingenioasă [fig.1-10], este obţinerea tensiunii Vcc şi parţial Vpp din semnalul de clock generat pe RTS, respectiv din înserierea cu aceasta a unei tensiuni de cca. 8.2V obţinută din semnalul TxD (X1-3). Limitarea amplitudinii semnalului CK (**C**loc**K**) la nivelul Vcc + 0.6V se face cu dioda D4, referinţa faţă de care se face identificarea nivelului logic al interfeţei RS232, fiind semnalul GND (X1-5) livrat de PC, respectiv pinul Vcc al conectorului SV1 (In Circuit Serial Programming). Diodele D3 şi D4 dublează de fapt diodele de protecţie existente pe intrarea fiecărui pin IO al microcontrolerului, fiind redundante dacă destinaţia programatorului este doar familia de microcontrolere PIC. Dacă se intenţionează programarea memoriilor eeprom seriale (I2C), montarea acestor diode este necesară. Când DTR este în 0 logic (tensiuni pozitive) tranzistorul T2 este în conducţie acţionând ca un stabilizator al nivelului amplitudinii semnalului de date (SD) la valoarea Vdd-0.6V. R2 este rezistenţa de sarcină pentru acest tranzistor, în unele variante ale acestui programator ea lipseşte, deoarece se presupune că interfaţa asigură limitarea curentului generat prin pinul DTR. Intoarcerea pe CTS este solicitată de programator ca feedback, pentru a testa nivelul semnalului SD la intrarea în programator. Când DTR trece în 1 logic (tensiuni negative) colectorul tranzistorului T2 este negativat atât faţă de bază aflată la +5V, cât şi faţă de emitor, tranzistorul fiind blocat. Tensiunea de alimentare a PIC-ului este asigurată din RTS când acest semnal se găseşte în 0 logic (tensiuni pozitive) prin D4, respectiv prin D3, când RTS este în 1 logic (are tensiuni negative) şi este menţinută la trecerea lui RTS în 1 logic (tensiuni negative) prin încărcarea condensatorului C1; limitarea amplitudinii la +5V se face de către dioda zenner D2. TxD generează în 0 logic (tensiuni pozitive), tensiunea de programare Vpp prin polarizarea tranzistorului T1, iar în perioada când TxD este 1 logic (tensiune negativă) asigură alimentarea PIC-ului prin D1. Tensiunea este menţinută cu condensatorul C2 pe perioda modificării nivelului logic al semnalului TxD în 1 logic. Protecţia joncţiunii bază emitor a tranzistorului T1 împotriva tensiunilor negative se face cu dioda D5. In momentul programării, RTS şi DTR trebuie să fie în 0 logic (tensiuni pozitive) şi să nu stea în 1 logic o perioadă prea lungă. Dacă atât RTS, DTR şi TxD stau în 1 logic (tensiuni negative) mai mult de 500mS, tensiunea de programare Vpp scade sub 8V, programarea fiind dezactivată .

Programatorul descris poartă denumirea simbolică JDM (după iniţialele realizatorului) şi este cel mai complet programator serial ce utilizeaza principiu NOPPP (**NO** **P**arts **P**IC **P**rogrammer) dar care asigură o tensiune de programare atât teoretic cât şi practic corectă. Există mai multe variante simplificate ale aceleiaşi scheme a căror principiu de funcţionare este dubios şi nenumărate alte variante cu tensiune de programare la limita inferioară, a caror utilizare nu este recomandată. Programele software care acceptă acest programator şi au fost testate de autor sunt:

http://www.ic-prog.com http://www.lancos.com

Concluzii: *programatorul serial este un programator de gabarit redus recomandat doar situaţiilor de programare pe teren când transportul unui programator paralel este incomod şi laptop-ul utilizat generează pe COM tensiuni suficient de mari. Utilizează integral tensiunea generată de interfaţa serială a calculatorului. Permite programarea microcontrolelor PIC şi a memoriilor eeprom seriale. Facilitatea ICSP trebuie utilizată cu grijă, PIC-ul trebuie să fie total separat de circuit în momentul programării.*

1.2.5 Programatorul de mare serie

Programatorul destinat producției de serie este identic din punct de vedere funcțional cu programatorul paralel sau serial descris anterior, şi utilizează de obicei un microcontroler supervizor şi un convertor de nivel pentru interfața serială de tipul MAX232 şi alimentare independentă, având în plus două facilități importante: programarea cu limite minime şi maxime ale tensiunii de alimentare şi generarea automată a semnăturii. Semnătura este un cod specific hexazecimal care conferă fiecărui cip unicitate, prin citirea acestuia producătorul poate identifica lotul, data fabricației produsului, etc. Multe programatoare care nu sunt declarate de serie au această funcție inclusă.

Programarea la limita tensiunilor de alimentare (modificarea valorii tensiunii de alimentare se face prin comandă software) verifică validitatea performanțelor declarate ale microcontrolerului în momentul programării, pe întreaga gamă a tensiunilor de alimentare, conform specificației tehnice. Dacă apare o pană, neobservabilă de altfel în condiții de alimentare cu tensiune nominală, microcontrolerul este inlocuit imediat ca urmare a unei pene de programare.

Un programator profesional ce acceptă un număr mare de eepromuri seriale şi paralele, microcontrolere Microchip şi Atmel, poate fi construit dacă se urmăresc îndeaproape indicațiile existente la: http://www.willem.org .

1.3 Utilizarea editorului şi a compilatorului

Faza inițială a scrierii programului debutează întotdeauna cu editarea textului liniilor de program. Pentru a evita abordarea limbajului de asamblare încă din faza de introducere, vom utiliza un compilator ce foloseşte un limbaj structurat înrudit cu pascal-ul dar mult mai intuitiv. Prezentarea acestuia se face pe larg în capitolul 2.

1.3.1 Editorul si mediul IDE (Integrated Development Environement)

Editarea propriuzisă se poate face cu orice program editor (notepad, wordpad, word) sau mai bine utilizând un editor profesional ce poate fi descărcat de pe WEB [10] numit PFE (CD:\editor\PFE). Dispune de toate facilitățile unui editor performant: căutare, înlocuire, setarea căutării în directoarea specificată, salvarea unui fişier backup ce conține modificările anterioare ale programului, etc. Editorul PFE are posibilitatea de a lansa în execuție compilatorul cu comanda *Execute* din meniul principal, urmată de *DOS Command to Window* pentru prima configurare şi compilare sau *Repeat DOS Command to Window* pentru compilările ulterioare. Utilizatorul trebuie să-şi editeze o filă batch (*.bat) care să conțină linia de comandă a compilatorului. Lansarea acestei file trebuie făcută din aceeaşi directoare în care se găseşte executabilul compilatorului iar rezultatul compilării va fi plasat tot aici. De exemplu:

D:\jal\bin\jal.exe -sD:\jal\lib D:\proiecte\test\%1.jal

Executabilul de mai sus se poate salva sub denumirea *jal.bat* în directorul d:\proiecte. In prealabil este necesară configurarea locului unde editorul caută fila *jal.bat*, în fereastra *Execute DOS Command and Capture Output* a editorului, (comanda *Execute* urmată de *DOS Command to Window*). Rezultatul execuției, cu comanda **jal nume** (unde *nume.jal* este fila text ce contine programul pe care l-am scris, filă editată în prealabil şi salvată în exemplul nostru, în fişierul d:\proiecte\test), va fi compilarea filei *nume.jal* şi obținerea filelor *nume.hex* respectiv *nume.asm*; compilatorul va căuta bibliotecile standard incluse în programul sursă în directoarea lib, dar în prealabil vor fi căutate bibliotecile create de utilizator în directoarea d:\proiecte\test. Dacă sunt utilizate numai bibliotecile standard, atunci opțiunea de căutare în biblioteci utilizator poate să dispară din linia de comandă. Nu uitați să precizați în aceeaşi fereastră PFE şi directoarea unde se găseşte executabilul *jal.bat* altfel rezultatul execuției va fi un mesaj de eroare. Cu o configurare corespunzătoare a PFE, în fereastra *Execute* urmată de *Launch Application,* se poate lansa în execuție şi programatorul favorit apelând executabilul corespunzător din directoarea unde programul software pentru programator a fost instalat. Astfel editorul se transformă uşor într-un mediu integrat de dezvoltare (**I**ntegrated **D**evelopment **E**nvironement) având soluționate trei elemente importante: editarea, compilarea şi programarea automată. Dacă utilizatorul doreşte să simuleze rezultatul compilării, acest lucru este posibil utilizând fila *nume.asm* împreună cu mediul MPLAB creat de Microchip sau cu comanda test.bat, o filă batch ce conține linia de mai jos:

D:\jal\bin\jal.exe -t -sD:\jal\lib D:\proiecte\test\%1.jal

unde opțiunea –t este folosită de simulatorul intern al compilatorului Jal.

Un alt editor realizat la nivel de amator este Jaledit [11] (CD:\editor\jaledit). Are aproximativ aceleaşi funcții ca şi PFE, cu câteva deosebiri:

- liniile editate sunt numerotate automat într-o fereastră din stânga ecranului,
- comentariile sunt colorate diferit în mod automat după terminarea editării rândului,
- lansarea compilării se face automat precizând doar directoarea în care se găseşte compilatorul, rezultatul compilării este plasat în aceeaşi directoare cu programul sursă,
- există posibilitatea generării automate a unui header care să conțină numele autorului şi data scrierii programului
- posibilitatea lansării (după o configurare prealabilă) a programatorului de microcontrolere
- posibilitatea setării dorite a culorilor pentru background şi text

Bineînțeles că există şi dezavantaje cum ar fi timpul lung de compilare, dublu decât în fereastră DOS. Un alt mediu interesant de editare-compilare-programare este Jal Command Center [12] (CD:\editor\jalcc) pe care sunteți invitați să-l descoperiți singuri !

1.3.2 Cum funcționează compilatorul Jal ?

Compilatorul poartă denumirea de Just Another Language [13] (adică "doar un alt limbaj") (CD:\tools\jal_compiler\) şi toate precizările următoare se referă la versiunea JAL 04.xx pentru WIN9x. Compilatorul Jal are o interfață de lucru în linia de comandă. Acelaşi compilator este disponibil pentru sistemul de operare DOS pentru windows (unde interfața DPMI este inclusă în sistemul de operare) şi pentru linux -386. Configurația hardware

minimală pentru a rula Jal este 486 (nu este necesar coprocesorul matematic) cu minim 4Mb RAM şi 10Mb memorie virtuală. Cu cât memoria (reală şi virtuală) este mai mare, cu atât este mai bine pentru utilizator. Codul sursă al compilatorului editat în C, este disponibil utilizatorului.

Compilatorul nu utilizează biblioteci compilate, întreaga fila sursă a unei aplicații este compilată odată. Acest lucru simplifică compilatorul şi permite analiza globală şi optimizarea, dar face ca procesul compilării să se desfăşoare mai lent. În urma unei compilări reuşite, compilatorul produce două file de ieşire. Numele de bază, fără extensii ale acestor file este acelaşi cu numele filei sursă prezente în linia de comandă. Prima filă de ieşire este de tipul *.*hex* şi conține copia hexazecimală a memoriei program a microcontrolerului. Aceasta filă poate fi utilizată în mod direct cu majoritatea programatoarelor de microcontrolere existente. A doua filă este de tipul *.*asm* şi conține fişierul asamblat al programului sursă. Această filă este foarte importantă, ea poate fi utilizată la verificarea codului generat şi pentru a face mici modificări sau corecții. Fila *.*asm* poate fi asamblată cu assemblerul standard Microchip (MPLAB/MPASM). Compilatorul Jal are deasemenea opțiuni pentru a seta diverse facilitați pentru determinarea erorilor.

1.3.3 Linia de comandă JAL

Linia de comandă DOS, conține numele filei sursă şi opțiuni. O opțiune începe cu "-", orice altceva este interpretat ca fiind o filă sursă. Rezultatul compilării este un fişier având acelaşi nume cu fila sursă dar cu extensia *.hex respectiv *.asm. Opțiunile liniei de comandă :

-t sau -tN : test

Opțiunea –t permite testarea programului compilat utilizând simulatorul încorporat. Opțional poate fi specificat numărul maxim de instrucțiuni (implicit este 10_000_000). Implicit testul nu se efectuează. In programul sursă trebuiesc utilizate instrucțiunile *pragma test assert* pentru determinarea valorii registrului testat, respectiv *pragma test done* pentru terminarea simulării.

-386 : funcționare pe calculatoare 386 - 486

Opțiunea – 386 trebuie folosită doar când se lucrează pe calculatoare cu resurse limitate. Este echivalentă cu –vz –cX. Pentru următoarele opțiuni literele mici setează iar cele mari resetează toate categoriile. (x setează, X resetează)

-c$: verifică

Optiunea –c comută verificarea on sau off. Implicit este -cxBP; setează toate verificările cu excepția memoriei şi a blocurilor de memorie. $ poate fi o secvență de :

```
b : verifică blocurile de memorie (memory Blocks)
p : verifică memoria globală (memory Pool)
a : verifică declarațiile valide interne (Assertions)
s : verifică utilizarea stack-urilor (Stack)
z : sterge memoria înainte de utilizare
```

-o$: optimizări

Optiunea –o comută optimizarea on sau off . Implicit este –ox: porneşte toate optimizările. $ poate fi o secvență de:

```
f : împachetarea expresiilor constante
    (Constants Folding)
```

```
r : micşorare a codului (Reduction)
s : reordonarea expresiilor (Tree Shape)
c : înlănţuirea call-urilor (Call chaining)
t : împachetarea expresiilor greşite (Trivial Expression)
d : înlăturarea codului ineficient (Dead Code Removal)
```

-s$: căutare

Optiunea -s adaugă directoarea $ listei de directoare în care sunt căutate filele incluse. Directoarele sunt căutate în ordine inversă: directoarea menţionată de prima opţiune -s este căutată ultima. Directoarea curentă este întotdeauna căutată prima. Este recomandat ca bibliotecile incluse în pachetul JAL să fie descărcate într-o subdirectoare **lib** şi întotdeauna să existe opţiunea **-slib** în linia de comandă.

-v$: detaliat (verbosity)

Opţiunea –v comută afişarea evoluţiei compilarii on sau off. Opţiunea este folositoare pentru detectarea şi înlăturarea erorilor (debugging) generate de compilator. Implicit este –vz : afişarea evoluţiei compilării. Activarea oricărei alte opţiuni duce la dezactivarea afişării paşilor compilării. $ poate fi o secvenţă de:

```
s : afişarea procesului compilarii în Scanner
p : afişarea procesului compilarii în Parser
o : afişarea procesului compilarii în Optimizator
q : afişarea procesului compilarii în sQuasher
r : afişarea procesului compilarii în Register allocation
c : afişarea procesului compilarii în Code generator
a : afişarea procesului compilarii în Assembler
t : afişarea procesului compilarii în simulator (Test)
z : afişarea evoluţiei diferiţilor paşi
```

comandă simplă de compilare a unei singure file sursă:

```
jal file
```

comandă complexă: caută în subdirectoarea \jal\lib, compilează pentru un 16F84_4, utilizează biblioteca jlib, fără optimizări, toate verificările fără zona de memorie, cu afişarea desfăşurării mecanismului optimizatorului:

```
jal -s\jal\lib 16f84_4 jlib file -oX -cxP -vo
```

Dacă utilizatorul doreşte să creeze o directoare specifică pentru un anume proiect şi să plaseze în acea directoare toate bibliotecile create de el, menţinând neschimbată structura directorului jal\lib, atunci linia de comandă este asemănătoare cu cea prezentată în 1.3.1 Descrierea pe larg a setului de instrucţiuni, operaţii matematice şi operatori, utilizate de compilator se face în capitolul 2.

1.4 Bootloader pentru microcontrolerul PIC16F87x

Microcontrolerele din seria PIC16F87x au o proprietate remarcabilă, aceea de a-şi putea modifica conţinutul memoriei program în timpul execuţiei programului propriuzis. Arhitectura structurată pe pagini de memorie, care este greoaie şi neplăcută utilizării curente este în această situaţie benefică deoarece permite protejarea zonei de memorie în care se găseşte programul principal (firmware-ul de bootloader), în timp ce zona rezervată programului utilizator poate fi încărcată/ştearsă de un număr impresionant de ori cu programul destinat aplicaţiei. Dacă o programare clasică prin ICSP durează câteva minute, şi necesită existenţa unui programator, încărcarea programului utilizator cu o rată de transfer de 19200bps prin bootloader, durează proporţional cu lungimea programului, între câteva secunde şi maxim două minute.

Ca **avantaj major** se menţionează aspectele:

- Folosirea unei conexiuni simple cu calculatorul pe una din interfeţele seriale disponibile în orice PC, conexiune care rămâne pe tot parcursul etapei de dezvoltare, nefiind necesară îndepărtarea ei la faza de test, ca în cazul programării prin ICSP, când circuitele aferente conectate pe pinii de programare trebuie să nu fie încărcate cu impedanţe mici pentru a realiza o programare corectă (deconectarea bootloaderului este obligatorie numai după un reset, când urmează faza de funcţionare independentă a aplicaţiei).
- Aplicaţia consumă un singur pin al microcontrolerului utilizat pentru transfer serial bidirecţional şi un hardware auxiliar atât de redus încât poate fi realizat pe un circuit imprimat simplu placat cu dimensiunile de 10x30mm, aplicaţia utilizatorului având un singur conector miniatură cu patru contacte (din care unul este cheia) pe care hardware-ul bootloaderului se conectează.
- Orice modificare ulterioară cerută de aplicaţie, înseamnă o simplă conectare a bootloaderului şi transferul programului utilizator.

Desigur că există şi **dezavantaje** ale utilizării bootloaderului:

- nu poate fi utilizată facilitatea de protecţie a memoriei program împotriva citirilor nedorite dacă se utilizează un bootloader. Corecţia acestei situaţii se face prin înscrierea clasică a programului în forma sa finală şi setarea corespunzătoare a fuzibilelor de protecţie, dacă este necesară protecţia firmware-lui înscris în PIC.
- Se consumă un pin al microcontrolerului a cărui destinaţie este doar bootloaderul (intrare-ieşire), rămânând disponibilă pentru utilizator doar funcţia de intrare de uz general a pinului respectiv. Microcontrolerele cu capsula de 40 de pini, dispun de suficiente rezerve hardware şi această situaţie când este necesar ultimul pin disponibil cu funcţie bidirecţională apare foarte rar.

Bootloaderul prezentat poartă denumirea generică Wloader [14] (CD:\tools\wloader). Wloader-ul se compune dintr-un ansamblu hardware [fig.1-11], firmware (wloader.asm), PCsoftware (wisp). El permite controlul inteligent al aplicaţiilor care încep de la adresa 0000 şi are o interfaţă de comandă DOS prin intermediul programului Wisp [15] (viespe). Firmware-ul loaderului ocupă 1Koctet de memorie în zona de vârf a memoriei, astfel cei 7 Kocteţi (PIC16F874/877) rămân disponibili utilizatorului. Se recomandă utilizarea lui cu microcontrolerele având mai mult de 4Kocteţi de memorie (PIC16F873/874/876/877) pentru a rămâne ceva memorie şi aplicaţiei utilizator. Schema electronică a bootloaderului din [fig.1-11], extrem de simplă, evidenţiază conexiunile Wloader-ului atât pe COM1

(conector cu 9 pini) cât şi pe COM2 (conector cu 25 de pini), utilizatorul având opţiunea de a le monta pe amândouă sau numai pe cel corespunzător mufei disponibile în calculatorul utilizat. Cablul de conexiune între conectorii de tip mamă şi plăcuţa de circuit imprimat nu va depăşi 1.5m, se poate utiliza cu succes o panglică cu 6 fire, trei active (TX, RX şi DTR) intercalate cu trei circuite de masă (GND). In acest mod de conexiune, perturbaţiile induse din exterior sunt foarte mici chiar pentru un cablu ceva mai lung.

Un semnal logic 0 (standard RS232) pe DTR (tensiuni pozitive) va deschide tranzistorul T1 (orice tip de tranzistor NPN), conectând linia de reset a PIC-ului (MCLR) la masă. Acest pin (MCLR) se conectează obligatoriu cu o rezistenţă de 4k7…10k la linia de +5V (Vcc) în circuitul utilizatorului, un nivel logic 1 pe MCLR menţinând în funcţionare normală PIC-ul, în timp ce un 0 logic îl resetează. Un semnal logic 1 pe linia DTR (tensiuni negative RS232) va bloca tranzistorul, protejând joncţiunea bază-emitor prin intermediul diodei D2 şi limitând curentul prin R1. Semnalul Tx este redus la limita de +5V prin R3, D1, R4, iar semnalul de întoarcere Rx este compatibil cu nivelul logic necesar pe interfaţa serială fiind mai mare de +3V (vezi CD:\PC_info\com.pdf şi capitolul 6), o simplă limitare a tensiunilor negative fiind suficientă prin intermediul rezistenţei R2. Pinul de transmisie/recepţie WLOAD se poate conecta la orice pin bidirecţional al microcontrolerului, împreună cu o rezistenţă de pull-up R5 de 47K. Utilizatorul va avea în vedere atunci când compilează fila *wloader.asm*, să menţioneze în program numele pinului utilizat fizic ca pin de comunicaţie, pentru a obţine o filă *wloader.hex* corectă. Această versiune de bootloader funcţionează perfect cu o gamă largă de calculatoare posedând interfeţe seriale variate de la laptop-uri 486 la calculatoare Pentium.

Cum funcţionează firmware-ul bootloaderului ? Loaderul pune instrucţiuni de tipul *goto* la adresele 0…2, care se activează în cazul unui reset. Instrucţiunile programului utilizator care ocupă în mod normal aceste adrese, sunt mutate într-o altă zonă din interiorul programului loader şi sunt executate înaintea saltului spre restul aplicaţiei utilizatorului care

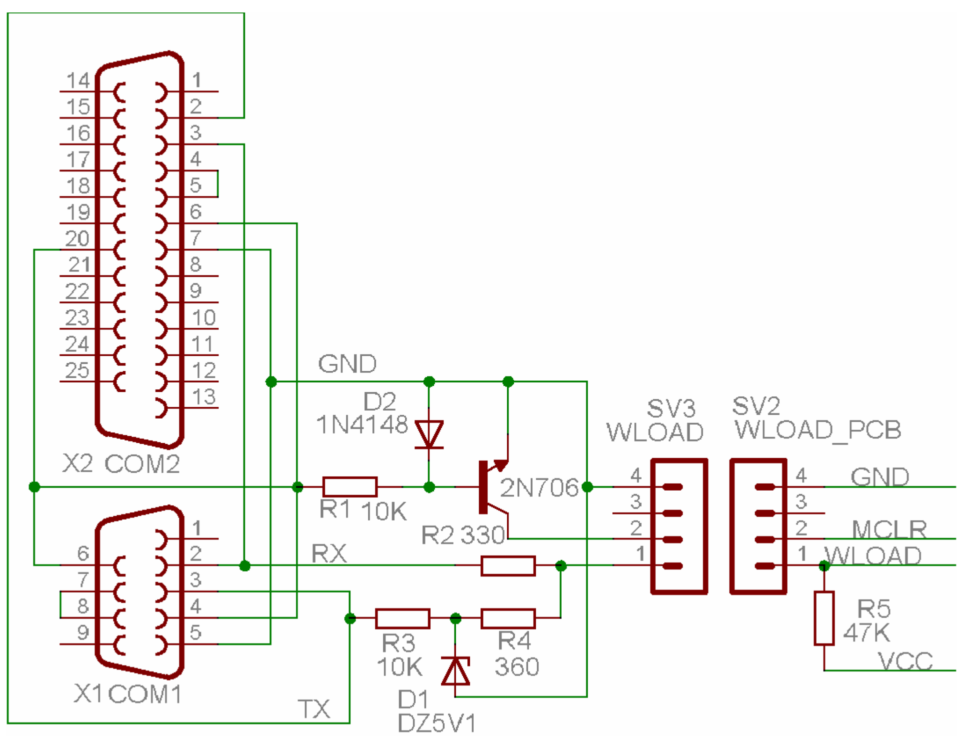


Fig.1-11 Bootloader pentru PIC17F87x, schema de principiu

începe cu adresa 3. Din acest motiv aplicația încărcată de loader poate fi structurată exact ca și un program de sine stătător care se programează într-un PIC16F877. Această șmecherie nu va funcționa dacă programul utilizator va încerca să utilizeze primele adrese pentru ceva mai inteligent, ca de exemplu generarea unei întârzieri:

```
$0000 reset_vector:      goto main
$0001 delay_12:          call $+1
$0002 delay_8:           call $+1
$0003 delay_4:           ret
$0004 interrupt_vector:  ...
```

Din fericire majoritatea compilatoarelor, inclusiv Jal, nu sunt atât de inteligente încât să poată genera un astfel de cod. Dacă utilizatorul îl scrie însă în assembler, atunci aplicația utilizator trebuie să conțină în primele trei adrese instrucțiunea NOP. Loaderul acceptă de la PC instrucțiuni de configurare a fuzibilelor. Acestea sunt grupate în PIC în registrul de 14 biți numit "configuration word". Pe calculator va trebui să fie instalat executabilul **wisp.exe** (CD\tools\wisp) care transferă o serie de comenzi bootloaderului. Comenzile pe care acesta le acceptă sunt:

BEEP	Indicație sonoră de succes sau eroare
BURN xxxx	specifică ID-ul (fuzibilul având patru caractere hexazecimale) care va fi programat în PIC, loaderul memorează acest cod în memoria flash
CHECK	Verifică dacă imaginea hexa a fost scrisă corect prin citirea ei din PIC şi compararea cu imaginea din PC
FLUSH	Se utilizează în combinație cu comanda LOG şi curăță fila log după fiecare scriere. Incetineşte procesul mai mult decât o face comanda LOG dar poate fi folositor pentru depistarea erorilor
FUSES spec	Specifică dacă cuvântul de configurare specificat va fi setat conform informației din fila hexa, (FUSES FILE), nu va fi setat (FUSES IGNORE), sau va fi setat cu o valoare specifică (FUSES value)
GO hex-file	Se execută o combinație de comanzi : ştergere, scriere, verificare şi rulare a programului încărcat în PIC
HEX	O comandă auxiliară pentru TALK, TERM, TTY, va arăta după fiecare caracter recepționat, valoarea hexa a caracterului respectiv în paranteze drepte
ID xxxx	Specifică valoarea identificatorului. Valoarea inițială este 0000. Se utilizează când mai multe PIC-uri sunt conectate pe acelaşi port serial şi fiecare PIC trebuie activat separat. In acest caz fiecare PIC trebuie programat cu un identificator diferit.
LAZY	Comută pe scriere lentă, dispozitivul va citi prima dată conținutul locației de memorie şi numai dacă noua valoare este diferită de cea veche, o va scrie. Valoarea inițială este de scriere normală.
LOG file	Fiecare activitate este memorată într-o filă log. Comanda este utilă pentru detectarea erorilor.
PORT p	Specifică pe ce port este conectat bootloaderul. Sunt recunoscute COM1, COM2, COM3, COM4. Implicit este COM1.
PORT b	Specifică viteza de comunicație. Viteza implicită este de 19200 bps.

PROTECT setting setting = [ON | OFF | IMAGE]
Controlează protejarea codului care poate fi setat: întotdeauna ON, întotdeauna OFF sau după cum este specificat în IMAGE. Wloader-ul nu poate scrie cuvântul de configurare, dar verifică dacă cuvântul încărcat se potriveşte cu cel programat.

READ hex-file Citeşte ținta şi scrie conținutul acesteia în fila hex indicată. Fila hex va conține intrări valide numai pentru acele locații de memorie care nu sunt în stare ştearsă (program 0x3FF, date 0xFF)

RUN Pune Pic-ul în funcționare după un reset, cu bootloaderul conectat

TALK Conversație cu dispozitivul: emulează o consolă TTY. Este utilă numai pentru detectarea erorilor.

TARGET target
target = [16f870 | 16f871 | 16f872 | 16f873 | 16f874 | 16f876 | 16f877]
Specifică dispozitivul țintă. Implicit este 16F877. Prefixul 16F poate fi omis.

TERM baudrate Emulează o simplă comunicație TTY cu viteza de comunicație specificată direct la portul serial al calculatorului. Viteza de comunicație trebuie să fie un întreg sau să conțină litera k pentru a indica sutele: 19200 sau 19k2

TEST Testează programabilitatea țintei cu 6 modele diferite de *pattern*-uri.

TIME Indică ceasul sistemului.

TTY baudrate Pune ținta în modul de rulare şi accesează transferul datelor spre ea. Se emulează o comunicație simplă TTY la rata de transfer indicată. Aceasta trebuie să fie un numar întreg sau să conțină litera k pentru a indica sutele: 300 sau k3; 19200 sau 19k2.

VERBOSE Arată execuția fiecărei operații pe ecran. Această comandă face ca procesul de programare să fie extrem de lent dar poate evidenția problemele de comunicație.

VERIFY hex-file
Verifică dacă conținutul țintei corespunde cu fila hexa indicată. Locațiile de memorie care nu sunt specificate în fila hexa nu sunt verificate.

WAIT milliseconds
Aşteaptă cel puțin numărul indicat în milisecunde. Interferențe cu sistemul de operare al calculatorului poate produce întârzieri mai lungi decât cele specificate.

WRITE hex-file
Scrie fila hexa indicată în microcontrolerul țintă. Locațiile de memorie care nu sunt specificate în fila hexa sunt păstrate cu valoarea originală .

Cum se utilizează bootloaderul? Pentru faza inițială, utilizatorul are nevoie de unul din programatoarele descrise anterior a căror funcționare corectă a fost testată, utilizând fie programul “flashing led” fie alt program asemănător de pe CD. Apoi, poate transfera în memoria PIC-ului direct fila *wloader.hex* (CD:\tools\wloader), utilizând programatorul preferat. Bootloaderul rezultat va funcționa numai cu un PIC16F877 având un cristal de cuarț de 20MHz, WLOAD [fig.1-11] fiind pinul E2, iar RES fiind echivalent cu MCLR. Pentru a asambla bootloaderul cu alte opțiuni specifice utilizatorului, trebuie ca mai întâi să fie instalat programul MPASM. MPASM este o parte a mediului IDE numit MPLAB [16] şi se găseşte pe site-ul Microchip. Apoi se despachetează pachetul *firmware.zip* aflat în CD:\tools\wloader\ într-o directoare goală şi pentru orice PIC16F87x cu memorie de 8K

rulând la 20MHz, se porneşte programul MPLAB, se apelează fila *wloader.asm* cu comanda *file* urmată de *open*, apoi *project* urmată de *build node*. Va apare fereastra MPASM în care este necesară completarea liniei *additional command line option* cu:

```
/dELCHEAPO /dXTAL=D'20'*MHz /dPIN=portd,2 /dBAUDRATE=D'19200'
```

Dacă utilizatorul foloseşte MPASM în mod DOS comanda va fi:

```
mpasm /dELCHEAPO /dXTAL=D'20'*MHz /dPIN=portd,2
/dBAUDRATE=D'19200' /dDEVICE_ID=B'00100110100000'
/dORIGIN=H'1C00' wloader
```

In ambele exemple se presupune că utilizatorul doreşte să conecteze bootloaderul la pinul d2 al PIC-ului. Este important ca în directoarea unde se găseşte *wloader.asm*, să existe şi fila de definire a microcontrolerului, *p16F877.inc*, respectiv ca mediul MPLAB să fie configurat pe tipul de microcontroler folosit, altfel vor apare erori la compilare. După lansarea comenzii *assemble*, dacă totul este în regulă, va rezulta fila *wloader.hex*. Mesajele de *warning* generate de MPASM pot fi neglijate. Atenţie, MPASM trebuie setat pe *case sensitivity = on*! De remarcat că mediul IDE trebuie setat pentru tipul de microcontroler pe care se va înscrie bootloaderul chiar dacă procesorul original pentru care a fost scrisă fila *wloader.asm* este PIC16F877. De exemplu dacă se doreşte înscrierea bootloaderului în PIC16F873, cu oscilator de 4MHz iar pinul de lucru va fi c0, în MPLAB, *options, development mode* se va seta acest tip de microcontroler. Comanda dată în fereastra *extra options* a MPASM va fi:

```
/dELCHEAPO/dXTAL=D'4'*MHz/dPIN=portc,0/dBAUDRATE=D'9600'/
dORIGIN=H'C00'
```

Opţiunea dELCHEAPO se referă la hardware-ul utilizat de bootloader, prezentat în [fig.1-11]. Viteza de comunicaţie a loaderului nu poate fi 19200bps decât cu cuarţ de 10 sau 20 MHz. Pentru 4Mhz alegeţi doar 9600 bps. Deoarece comanda implicită a utilitarului wisp (CD:\tools\wisp) este de 19200bps, dacă se utilizează o altă viteză, aceasta va fi specificată în linia de comandă **wisp** (comandă dată într-o fereastră DOS sub WINDOWS, pe PC) cu comanda PORT :

```
wisp port com2 port 9600 fuses ignore go blink.hex
```

Această comandă arată că wloader-ul este conectat pe portul COM2, viteza de comunicaţie este de 9600bps, se ignoră modificările fuzibilelor ID ce apar în fila hex şi se execută programul *blink.hex*. Rezultatul vizibil al acestei execuţii în PIC, va fi înscrierea filei *blink.hex* în PIC şi pâlpâirea LED-ului corespunzător setărilor din programul *blink.jal*.

Pentru a compila wloader-ul pentru PIC-uri ce au mai puţin de 8Kocteţi de memorie, punctul de origine al codului va fi setat la 1Koctet sub limita superioară maximă a memoriei şi fuzibilul cu rol de protecţie al blocului respectiv de memorie trebuie activat pentru protecţie. De exemplu, pentru PIC16F871 (2K code) comanda va fi:

```
mpasm /dDEVICE_ID=B'00110100100000' /dORIGIN=H'0400' wloader
```

Bootloaderul dezactivează automat funcțiile analogice ale microcontrolerului pentru a face posibilă utilizarea portului A ca şi intrări digitale. Imediat după ce a fost încărcat, bootloaderul setează acest port cu condițiile inițiale (funcțiile analogice activate, portul A nu poate fi utilizat ca intrări digitale). Sunt furnizate două programe de test scrise în JAL, un led care pulsează cu frecvența de 1 Hz, (*blink.jal*) care, după compilare poate fi rulat pe microcontroler cu comanda:

```
wisp fuses ignore go blink
```

respectiv un program care generează mişcarea unui led stins într-un şir de 32 de led-uri conectate pe toate ieşirile PIC-ului, mai puțin pinul E2 (*walk.jal*).

1.5 Microcontrolere Microchip Flash din seria midrange

1.5.1 Portretul robot al microcontrolerului flash Microchip midrange

Fiecare microcontroler abordat în aplicațiile prezentate aici, dispune de o documentație laborioasă editată în mai multe versiuni ale căror codificare (DSxxxxx) au extensiile A, B sau C. Deoarece rolul acestei cărți nu este în totalitate acela de a traduce o documentație din limba engleză, documentația originală a producătorului trebuie citită în detaliu, versiunile B sau C fiind cele în care erorile au fost corectate, dar în care se omit chestiuni esențiale considerate cunoscute şi care sunt prezentate în variantele inițiale A. Eratele care apar frecvent trebuiesc de asemenea citite. Ele se găsesc pe *site*-ul producătorului: http://www.microchip.com. Cea mai detaliată prezentare a microcontrolerului flash este făcută în DS30445A pentru PIC16C84, acesta fiind primul cip flash produs de Microchip şi care nu se mai fabrică. Celelalte documentații utile sunt:

PIC16F8x	- DS30430C
PIC16F84a	- DS35007A
PIC16F62x	- DS40300B
	- DS80047B
PIC16F7x	- DS30325A
	- DS80099A
PIC16F87x	- DS30292A
	- DS30292B
	- DS30292C
	- DS30925B
PIC16C84	- DS30189D
PIC12F675	- DS41190A
Midrange Manual	– DS32023A

Nu intrați în panică ! Fiecare documentație de mai sus conține cel puțin 200 de pagini.

1.5.2 Arhitectura internă

Microcontrolerele *flash* sunt declarate de producător ca fiind microprocesoare cu arhitectură RISC (**R**educed **I**nstruction **S**et **C**omputer) adică având un număr redus de instrucţiuni (36). Cu toate acestea, utilizatorul începător va avea destule bătăi de cap cu memorarea acestor instrucţiuni, deşi mnemonicele acestora sunt intuitive. Setul de instrucţiuni şi descrierea detaliată a acestora se găsesc în fila de catalog a fiecărui microcontroler. Un lucru pozitiv este faptul că fiecare instrucţiune durează un singur ciclu maşină (mai puţin instrucţiunile *call* şi *goto* care durează doi tacţi maşină) deci este uşor de determinat timpul consumat de o rutină sau o porţiune de program printr-o simplă numărare a instrucţiunilor. Elaborarea unor rutine de comunicaţie serială prin software este dificilă fără această facilitate. Viteza de operare a acestor microcontrolere este de numai 20MHz (un punct negru acordat producătorului) însă suficient pentru majoritatea aplicaţiilor comune.

Memoria program a familiei Microchip mid-range, variază de la 512 octeţi la 8Kocteţi, memoria RAM de la 36 de octeţi la 368 de octeţi iar memoria internă EEPROM de la 64 de octeţi la 256 de octeţi.
Toată familia dispune de o structură de bază având:

- maxim 15 surse de intreruperi interne şi externe,
- stivă hardware de 8 nivele,
- adresare directă, indirectă şi relativă a memoriei utilizând organizarea pe pagini pentru memoria program, bancuri de memorie pentru memoria utilizator şi regiştrii cu funcţii speciale, având posibilitatea protejării totale sau parţiale a paginilor de memorie,
- POR (**P**ower **O**n **R**eset) - facilitate de deosebire şi tratare adecvată a diverselor surse de reset a microcontrolerului:
    - reset din alimentare
    - reset din operare normală pe MCLR\ (**M**aster **CL**ear **R**eset)
    - reset din modul SLEEP (cu consum de curent redus) pe MCLR\
    - reset datorat WDT (**W**atch **D**og **T**imer)
    - “trezire” datorată WDT din modul SLEEP
    - BOR (**B**rown **O**ut **R**eset) – resetează microcontrolerul dacă alimentarea acestuia scade sub 3.7…4.4V (tipic 4V)
- PWRT (**P**o**W**e**R** up **T**imer) şi OST (**O**scillator **S**tartup **T**imer) facilităţi de întârziere la pornire cu o cuantă de timp fixă de 72mS, respectiv o întârziere de 1024 de tacţi oscilator, pentru a preîntîmpina startarea defectuoasă a programului datorată utilizării surselor de alimentare cu viteză de stabilizare mică, şi a aştepta intrarea în regim a semnalelor tranzitorii,
- WDT este un oscilator intern RC care are rolul de “câine de pază” cu durată fixă tipică de 18mS, dar care poate fi asignat unui postscaler (un registru numărător de 8 biţi) care este utilizat “la comun” cu registrul TMR0 şi pentru care devine prescaler. Astfel durata maximă de pază este de 7…33mS (variaţia minim-maxim a oscilatorului intern RC) înmulţită cu valoarea maximă a postscalerului (1:128), deci între 0,896…4,224 S. *Observaţi că s-au luat în calcul limitele extreme de variaţie a oscilatorului prezentate în capitolul “Electrical characteristics” a fiecărei documentaţii şi nu valoarea tipică de 18mS. Utilizatorul poate întotdeauna să efectueze calculele cu această valoare tipică însă trebuie să fie pregătit pentru aprecierea corectă a marjelor de eroare pe*

care le poate obține în practică datorită variației tensiunii de alimentare şi a temperaturii ambiante.

- Posibilitatea reducerii consumului de alimentare în regimul SLEEP
- Selectarea tipului de oscilator de tact utilizând programarea unor fuzibile corespunzătoare odată cu transferarea memoriei program. Sunt posibile următoarele tipuri de oscilatoare:
    - Oscilator extern
    - Oscilator RC intern (unele PIC-uri dețin şi posibilitatea calibrării acestuia utilizând informația din registrul intern OSCAL) sau extern
    - Oscilator cu cuarț sau rezonator XT, cuarț de mică putere LP, sau de frecvență ridicată HS
- Posibilitatea programării în circuit utilizând 5 semnale: Vcc (tipic +5V), Vpp (se aplică pe MCLR\ şi este tipic de 13.5V), tact (PGC-**P**ro**G**ramming **C**lock, RB6), data (PGD-**P**ro**G**ramming **D**ata, RB7) şi masa electrică (GND) pentru modul de programare HVP (**H**igh **V**oltage **P**rogramming), respectiv acelaşi număr de pini de date şi clock, dar cu tensiune VPP = 5V aplicat pe pinul RB3 pentru modul de programare LVP (**L**ow **V**oltage **P**rogramming). Doar ulimele generații de microcontrolere dispun de modul de programare LVP. Ca să funcționeze, LVP necesită o setare inițială a fuzibilelor prin HVP, această setare se face de obicei din fabrică. După programare LVP, pinul RB3 se conectează la masă, fie direct, fie printr-o rezistență de 4K7…10K.
- ICD (**I**n **C**ircuit **D**ebugger) – facilitate prezentă doar la seria PIC16F87x, utilă împreună cu suportul hardware ICD şi programul prezent în MPLAB, permite depistarea erorilor în programul sursă prin înserarea a maxim 5 *breakpoint*-uri (puncte de întrerupere). Prin inspectarea valorilor regiştrilor, utilizatorul poate depista sursa de erori cu condiția ca *breackpoint*-urile să fie inserate în bucla de program cu funcționare defectuoasă. Există şi alte metode de depanare a programelor foarte lungi, cea mai la îndemână fiind metoda extinderii de la simplu la complex, prin care rutinele sunt implementate/modificate/verificate individual pe un PIC funcționând cu un bootloader, după care se testează înlănțuirea lor corectă în programul final şi transferul acestuia pe microcontrolerul destinat aplicației.

Facilitățile oferite de perifericele înglobate în aceste microcontrolere sunt:

- Timer0 – timer (temporizator) şi numărător de 8 biți având un prescaler de 8 biți (1:1, 1:2,… 1:256) utilizat în mod comun cu WDT. Prescaler-ul este un simplu registru numărător.
- Timer1 – timer şi numărător de 16 biți cu prescaler (1:1, 1:2,…1:8) poate fi incrementat şi în starea de SLEEP a microcontrolerului (stare de consum redus).
- Timer2 – timer şi numărător de 8 biți cu registru de perioadă de 8 biți, prescaler (1:1, 1:4, 1:16) şi postscaler (1:1, 1:2, 1:4,… 1:16).
- Unul sau două module de captură, comparare sau PWM (**P**uls **W**idth **M**odulation – modulație în durată cu lărgime de puls); captura pe 16 biți cu rezoluția maximă de 12,5 nS, compararea pe 16 biți cu rezoluția maximă de 200nS, rezoluția maximă a PWM- ului este de 10 biți *(rezoluția şi frecvența maximă este de 78kHz/8biți la 20MHz tact oscilator, aceasta nu înseamnă că nu se pot obține frecvențe mai mari cu rezoluții mai mici).*

- Convertor AD multicanal (5 sau 8 canale) de 10 biți (PIC16F87x, PIC12F675, PIC16F676), 8 biți (PIC16F7x) sau unul/două comparatoare multifuncționale cu opt moduri distincte de utilizare (PIC16F62x, PIC16F87xA, PIC12F675, PIC16F630/676).
- Referință de tensiune cu rezoluție de 4 biți în domeniul 0…3.125V sau 1.25…3,75V (PIC16F628 la VCC = 5V).
- SSP (**S**ynchronous **S**erial **P**ort-port serial sincron) cu SPI funcționând în mod stăpân (master mode) şi I2C stăpân-sclav (master/slave). Această facilitate este extrem de utilă la interfațarea rapidă şi fără mari probleme software a convertoarelor AD cu interfață serială. Conectarea memoriilor EEPROM externe sau a altor periferice I2C pe bus-ul I2C al PIC-ului este de asemenea posibilă.
- USART (**U**niversal **S**ynchronous **A**synchronous **R**eceiver **T**ransmitter) cu detecția adresei pe 9 biți, este un modul extrem de valoros permițând transmisia asincronă full duplex (bidirecțională) cu viteze de până la 1Mbps.
- PSP (**P**aralel **S**lave **P**ort-port paralel sclav) de 8 biți cu control extern RD\ (read), WR\(write) şi CS\(chip select). Permite conectarea paralelă la orice sistem microprocesor.

Portretul robot fiind terminat putem să-l sintetizăm în [fig.1-13]. Structura internă conține:

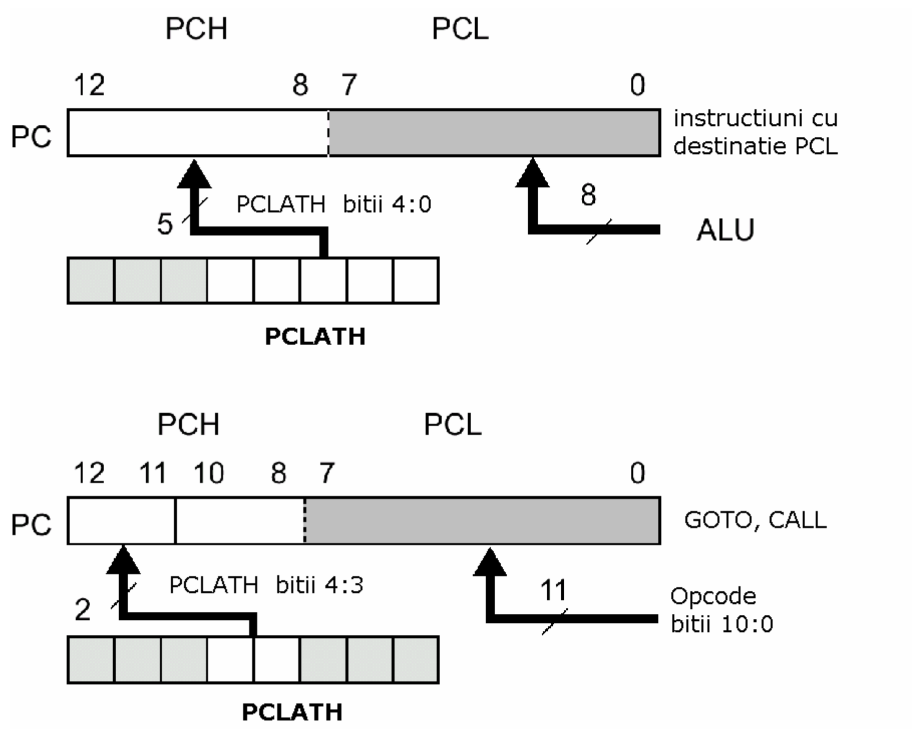


Fig.1-12 Incărcarea numărătorului de program în diferite situații

- Registrul numărător de program (*program counter PC*) împreună cu stiva (*8 level stack*) este ansamblul care memorează poziția instrucțiunii în curs. PC are dimensiunea de 13 biți şi este împărțit în două zone [fig.1-12] :
  - PCL (*program counter low*) un registru de 8 biți în care este permisă atât citirea cât şi scrierea, şi unde se plasează rezultatul operațiilor efectuate în unitatea aritmetico-logică (*ALU*).

- PCH (*program counter high*) un registru de 5 biţi în care informaţia este înscrisă numai printr-un registru tampon numit PCLATH. Transferul celor 5 biţi se face normal păstrîndu-se poziţia lor şi în registrul PCH, dacă a avut loc o scriere standard în PCL [**fig.1-12** sus], biţii 3 şi 4 din PCLATH devin biţii de offset 12 şi 11 din PC în cazul unei instrucţiuni GOTO compuse sau CALL, utilizate pentru citirea unui tabel din memoria program [**fig.1-12** jos] .

PC este salvat în stivă de fiecare dată când are loc o instrucţiune CALL sau o întrerupere cauzează lansarea unui program ramificat. Stiva este citită şi numărătorul de program reîncărcat cu valoarea salvată, după orice instrucţiune RETURN, RETLW sau RETFIE. Este important de reţinut că stiva este un buffer circular, dacă în ea s-a scris de 8 ori, a 9-a scriere va şterge poziţia întâia a stivei, a 10-a scriere suprascrie poziţia a doua şamd. De observat că nu există operaţii de tip *push* şi *pop* pentru operaţii cu stiva (ca la Z80) şi nici un bit al registrului STATUS nu semnalizează depăşirea stivei.

Depăşirea stivei (*stack owerflow*) poate avea loc dacă nu se utilizează cu grijă procedurile sau funcţiile imbricate, când într-o procedură se apelează o funcţie sau o altă procedură, ce conţine la rândul ei o altă procedură, etc. Singura modalitate acceptată de jal, fără a "umfla" stiva la apelarea înlănţuită a procedurilor, este plasarea procedurii chemate pe ultima linie a procedurii curente. Astfel rezultatul compilării va fi un *goto* în loc de *call*. Rezultatul depăşirii stivei este întoarcerea într-o altă zonă a programului decât cea din care s-a plecat, dintr-o eroare a conţinutului *program counter*-ului. Compilatorul JAL va sesiza depăşirea stivei şi o va raporta ca eroare la faza de asamblare.

- O zonă SRAM (Static Row Address Memory) şi o zonă de memorie program de tip FLASH ce variază ca dimensiuni de la microcontroler la microcontroler aşa cum este prezentat în cap.1.5.3. Din punct de vedere fizic memoria SRAM face parte din zona de memorie destinată regiştrilor cu funcţii speciale.
- FSR, un registru de adresare indirectă a memoriei. Adresarea indirectă este modalitatea cea mai rapidă de accesare a memoriei comparativ cu adresarea directă [fig.1-13].
- Memorie EEPROM nevolatilă. Pentru scrierea în memoria EEPROM internă, utilizatorul trebuie să respecte un algoritm fix precizat de producător.

O observaţie esenţială se referă la registrul de configurare *Configuraţion Word* (adresa 2007h) care se programează odată cu înscrierea microcontrolerului. Acesta este organizat pe 14 biţi, o parte din funcţiile biţilor sunt comune pentru majoritatea microcontroleror în discuţie, o altă parte sunt specifice numai anumitor microcontrolere. Dacă acest cuvânt de configurare nu este setat în concordanţă cu schema hardware, este posibil ca aplicaţia să nu funcţioneze deloc deşi restul programului este perfect. De exemplu pentru PIC16F628 acest registru are formatul următor:

bit13 bit0

CP1	CP0	CP1	CP0	-	CPD	LVP	BODEN	MCLR	FOSC2	PWRTE	WDTE	FOSC1	FOSC0

CP1: CP0 sunt biţii de protecţie ai memoriei program
CPD este bitul de protecţie al memoriei de date
LVP este bitul de setare al programării cu tensiune redusă

BODEN setează detecția scăderii tensiunii de alimentare sub limita de 4V
MCLRE setează tipul de reset al microcontrolerului
PWRTE setează temporizatorul la alimentarea microcontrolerului
WDTE este bitul care pornește câinele de pază
FOSC2:FOSC0 sunt trei biți ce setează tipul de oscilator (LSB)

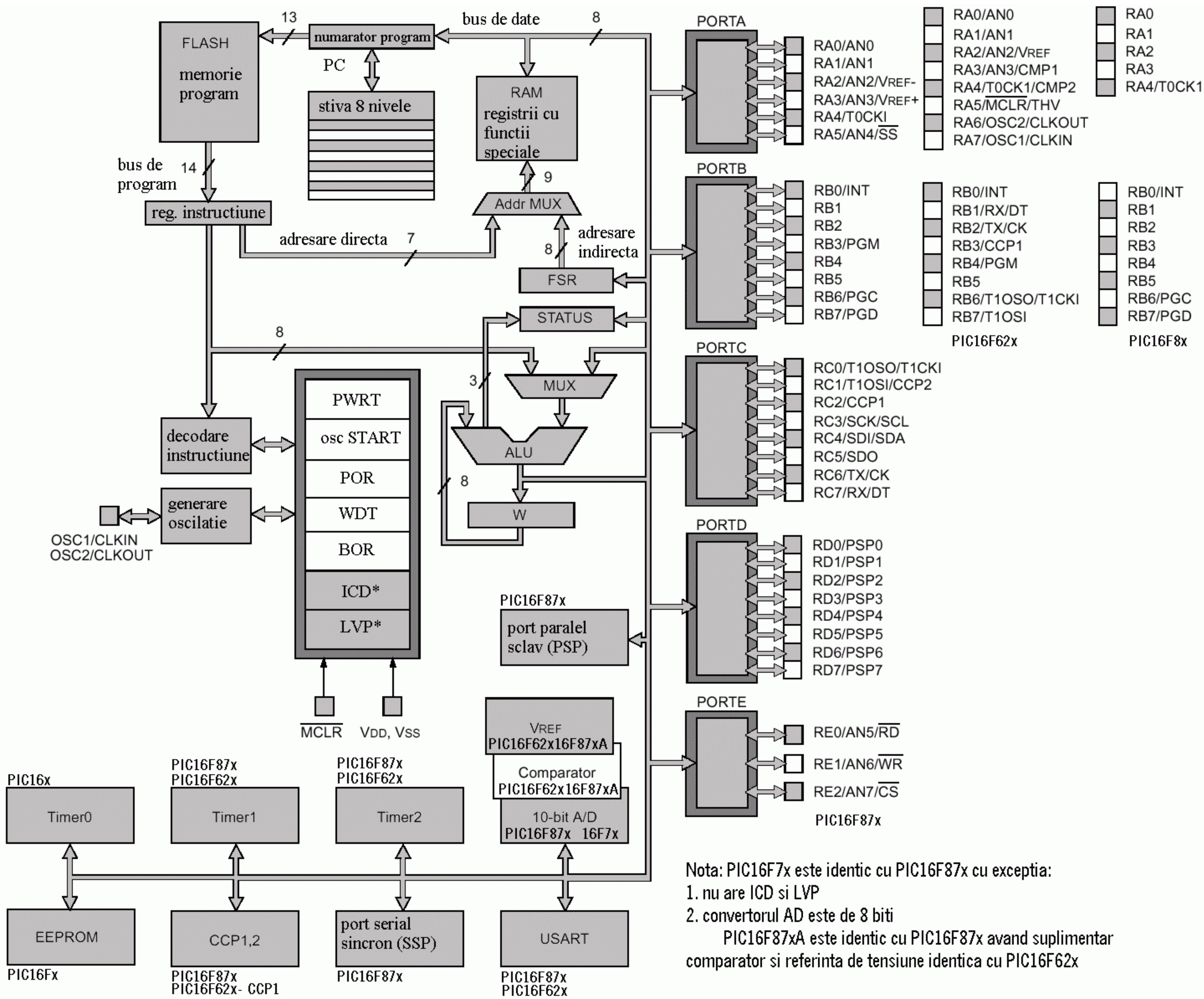


Fig.1-13 Portretul robot al microcontrolerului midrange-Microchip

De exemplu, setarea eronată a unui oscilator extern cu cuarț din FOSC2:FOSC0, când acesta lipseşte fizic, va duce la lipsa semnalului de tact şi la prezența unui microcontroler "mort".

Principalele deosebiri între câteva dintre tipurile de microcontrolere flash din familia mid-range, sunt prezentate sintetic în tabelul următor:

	flash	**ram**	**eeprom**	**adc**	**cmp**	**special**	**IO**	**serial**	**pwm**	**Osc[MHz]**
16F83	512	36	64	-	-	(2)	13	-	-	10
16F84	1K	68	64	-	-	(2)	13	-	-	10
16F84A	1K	68	64	-	-	(2)	13	-	-	20
16F627	1K	224	128	-	2	(3)	16	Usart	1x10bit	20
16F628	2K	224	128	-	2	(3)	16	Usart	1x10bit	20
12F675	1K	64	128	4x10bit	1	(5)	6	-	-	20
12F629	1K	64	128	-	1	(5)	6	-	-	20
16F630	1K	64	128	-	1	(5)	12	-	-	20
16F676	1K	64	128	8x10bit	1	(5)	12	-	-	20
16F70/870	2K	128	64	5/8x 8/10	(4)	(1)	22	Usart/i2c/spi	2x10bit	20
16F71/871	2K	128	64	8/8x 8/10	(4)	(1)	33	Usart/i2c/spi	2x10bit	20
16F72/872	2K	192	128	5/8x 8/10	(4)	(1)	22	Usart/i2c/spi	2x10bit	20
16F73/873	4K	192	128	5/8x 8/10	(4)	(1)	22	Usart/i2c/spi	2x10bit	20
16F74/874	4K	192	128	8/8x 8/10	(4)	(1)	33	Usart/i2c/spi	2x10bit	20
16F76/876	8K	368	256	5/8x 8/10	(4)	(1)	22	Usart/i2c/spi	2x10bit	20
16F77/877	8K	368	256	8/8x 8/10	(4)	(1)	33	Usart/i2c/spi	2x10bit	20

(1) BOR, 1xTmr0-8 bit, 1xTmr1-16 bit, 1xTmr2-8bit, 1xWDT, posibilitatea citirii şi a scrierii memoriei flash pentru pic16f87x aflat în funcționare normală (nu numai în faza de programare)
(2) 1xTmr0-8bit, 1xWDT, MCLR extern, oscilator extern
(3) BOR, 1xTmr0-8 bit, 1xTmr1-16 bit, 1xTmr2-8bit, 1xWDT, MCLR intern sau extern, oscilator intern sau extern de tip RC
(4) PIC16F87xA dispune de toate facilitățile lui PIC16F87x şi de comparatorul şi referința de tensiune a lui PIC16F62x
(5) BOR, 1xTmr0-8 bit, 1xTmr1-16 bit, 1xWDT, MCLR intern sau extern, oscilator intern sau extern de tip RC

1.5.3 Organizarea memoriei

Toate microcontrolerele flash mid-range au vectorul de reset la adresa 0h şi vectorul de întrerupere la adresa 04h. Adică programul principal va începe întotdeauna de la adresa 0h în timp ce tratarea întreruperii se va face cu rutina spre care dirijează *call*-ul

memorat la adresa 04h. Organizarea memoriei de date este pe maxim 4 bancuri de memorie notate de la *bank0* la *bank3*, accesarea bancului dorit se poate face fie cu biții *RP0* şi *RP1* în cazul adresării directe, fie cu bitul *IRP* în cazul adresării indirecte (în conjuncție cu valoarea memorată în registrul *fsr – file select register*), aceşti biți se găsesc în registrul STATUS. Exemplul din **fig.1-14** arată modul de adresare indirectă în microcontrolere cu 4 bancuri de memorie. Presupunând că:

- registrul cu adresa 05h (porta) are valoarea 10h
- registrul cu adresa 06h (portb) are valoarea 0Ah
- se încarcă valoarea lui 05h în registrul FSR, o citire a registrului INDF va returna acum valoarea 10h
- se incrementează FSR cu 1 (FSR = 06h), o nouă citire a INDF va returna 0Ah

Citirea indirectă a registrului INDF va produce valoarea 0h. Scrierea indirectă în registrul INDF va duce la neexecutarea nici unei instrucțiuni deşi registrul STATUS poate să se modifice. Un program interactiv care explică excelent ce se întâmplă în memoria PIC-ului se găseşte în [18].

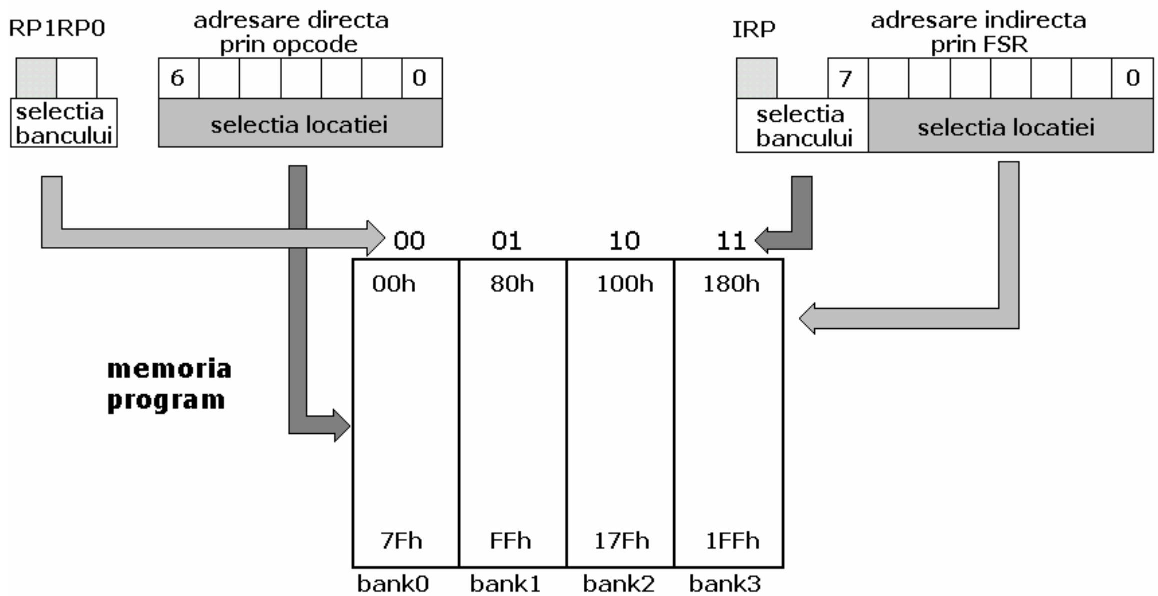


Fig.1-14 Adresarea directă şi indirectă

Harta memoriei celor patru bancuri diferă de la microcontroler la microcontroler, o parte din regiştrii cu funcții speciale sunt comuni tuturor microcontrolelor, o altă parte (în principiu cei care se referă la funcțiile analogice şi întreruperile acestora, sau la regiştrii hardware de comunicație) diferă după cum aceşti regiştrii sunt implementați fizic sau nu. Producătorul a ales un mod cel puțin ciudat de a "amesteca" pozitia regiştrilor cu funcții speciale cu cei de uz general, care sunt din punct de vedere ai utilizatorului regiştrii SRAM volatili, în care se pot stoca date atâta timp cât microcontrolerul este alimentat. Resetarea microcontrolerului duce la pierderea acestor date. Spre deosebire de memoria RAM, memoria EEPROM internă este nevolatilă, conținutul acesteia rămâne neschimbat şi după resetarea microcontrolerului.

In concluzie, microcontrolerul PIC dispune de memorie program FLASH, unde este stocat programul utilizatorului, memorie SRAM format din regiştri de uz general unde

sunt memorate datele aflate în perpetuă schimbare şi memorie EEPROM unde sunt stocate date pe termen lung. Unele microcontrolere permit stocarea datelor şi în memoria FLASH însă numărul de înscrieri garantat al acesteia este sub 10.000 de cicluri, spre deosebire de memoria EEPROM care are o rată de înscriere garantată de 100.000 de cicluri.

1.5.4 Regiştrii cu funcţii speciale

Numărul de regiştrii cu funcţii speciale este mai mare în arhitectura microcontrolelor cu 40 de pini deoarece şi resursele hardware sunt mai numeroase. Cu toate acestea există un număr de regiştrii de bază care sunt aceiaşi în toate microcontrolere flash, diferind doar adresa şi uneori denumirea acestora. Utilizatorul găseşte această informaţie în capitolul "Memory organization" în tabelul "PICxxx register file map" din fila de catalog a fiecărui microcontroler. Regiştrii sunt distribuiţi în toate cele patru bancuri de memorie, unii au adresă redundantă în toate bancurile de memorie: **status, pcl** (*program counter latch*), **intcon, pclath** sau doar în unele perechi de bancuri (bank0 şi bank2 respectiv bank1 şi bank3): **tmr0, option_reg, portb, trisb**. Pentru a accesa aceşti regiştrii este obligatoriu ca să aibă loc în prealabil setarea paginii de memorie corespunzătoare, prin adresare directă sau indirectă. O încercare de a sintetiza poziţia şi rolul regiştrilor cu funcţii speciale pentru microcontrolerele flash midrange este prezentată în [fig1-15, fig.1-16]. Se observă următoarele categorii de regiştrii:

- Regiştrii comuni tuturor microcontrolelor flash midrange : *TMR0, PCL, STATUS, FSR, PORTA, TRISA, PORTB, TRISB, PCLATH, INTCON.* Deşi apare ca un registru, *Indirect addressing* nu este implementat fizic, fiind utilizat doar pentru adresarea indirectă. Aceşti regiştrii reprezintă întreaga resursă internă a primului microcontroler flash produs de Microchip, PIC16C84 (PIC16F84)
- Regiştrii specifici funcţiilor analogice sunt: *CMCON, VRCON,* pentru setarea configuraţiei comparatoarelor şi a referinţei interne de tensiune în PIC16F62x, PIC16F87xA şi *ADRESH, ADRESL, (*sau *ADRES) ADCON0, ADCON1* pentru configurarea şi citirea rezultatului conversiei AD de 8 biţi (PIC16F7x) respectiv de 10 biţi (PIC16F87x, PIC12F675)
- Regiştrii porturilor de intrare-ieşire suplimentare: *PORTC, TRISC, PORTD, TRISD, PORTE, TRISE*, aceşti regiştrii sunt disponibili fizic numai pentru anumite tipuri de împachetare (capsule)
- Regiştrii asociaţi ai timerului 1: *TMR1L, TMR1H, T1CON*
- Regiştrii asociaţi ai timerului 2: *TMR2, T2CON, PR2*
- Regiştrii asociaţi ai modulului comparare/captură/pwm: *CCPR1L, CCPR1H, CCP1CON, CCPR2L, CCPR2H, CCP2CON*
- Regiştrii asociaţi accesului la memoria eeprom şi memoria flash: *EEDATA, (PMDATA) EEDATH, (PMDATH), EEADR, (PMADR), EEADRH, (PMADRH), EECON1, (PMCON1), EECON2 (*memoria flash nu este disponibilă în faza de rulare a programului numai în seria PIC16F87x. Acest lucru înseamnă că numai această familie este capabilă de a-şi modifica o zonă a memoriei program în timpul rulării programului înscris într-o zonă protejată la scriere a memoriei flash.)
- Regiştrii specifici modulului USART (Universal Synchronous Asynchronous Receiver Transmiter) : *TXREG, RCREG, RCSTA, TXSTA, SPBRG*
- Regiştrii asociaţi modulului MSSP (Master Synchronous Serial Port) : *SSPSTAT, SSPCON, SSPCON2, SSPBUF, SSPADD* (numai pentru PIC16F87x şi PIC16F7x)

- Diverşi regiştrii utilizaţi de întreruperi: *PIR1, PIR2, PIE1, PIE2* împreună sau nu cu registrul *INTCON*
- Regiştrii pentru funcţiile speciale de alimentare: *PCON*
- Regiştrii haşuraţi nu sunt disponibili pentru utilizator (fie nu există fizic, fie sunt rezervaţi)

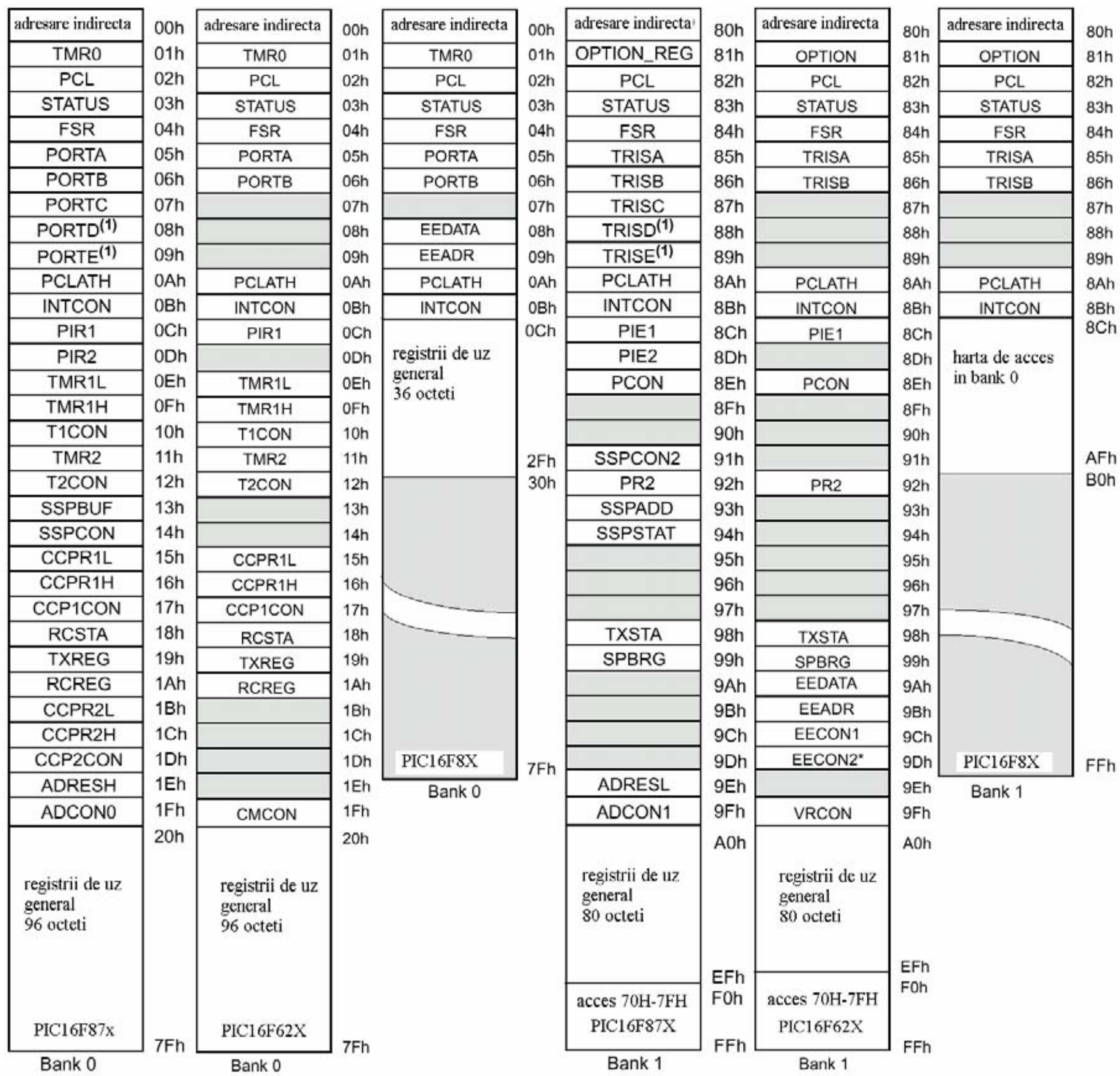


Fig. 1-15 Regiştrii PIC în bank0 şi bank1

Numărul mare de regiştrii cu funcţii speciale nu trebuie să-l sperie pe începător. Este esenţial ca abordarea acestora să se facă metodic, motiv pentru care, după ce s-a optat pentru tipul de microcontroler, (ideal este pentru început să se lucreze cu un microcontroler cu resurse limitate ca PIC16F84 sau mai elegant PIC16F628 sau PIC12F675) se va lista întreaga documentaţie existentă pe CD sau WEB, referitoare la microcontrolerul respectiv. Proiectarea schemei electronice (sau înţelegerea unui proiect eleborat de altcineva) se va face cu documentaţia deschisă pe masă. O parcurgere prealabilă "în viteză" a documentaţiei microcontrolerului simplifică foarte mult existenţa tuturor.

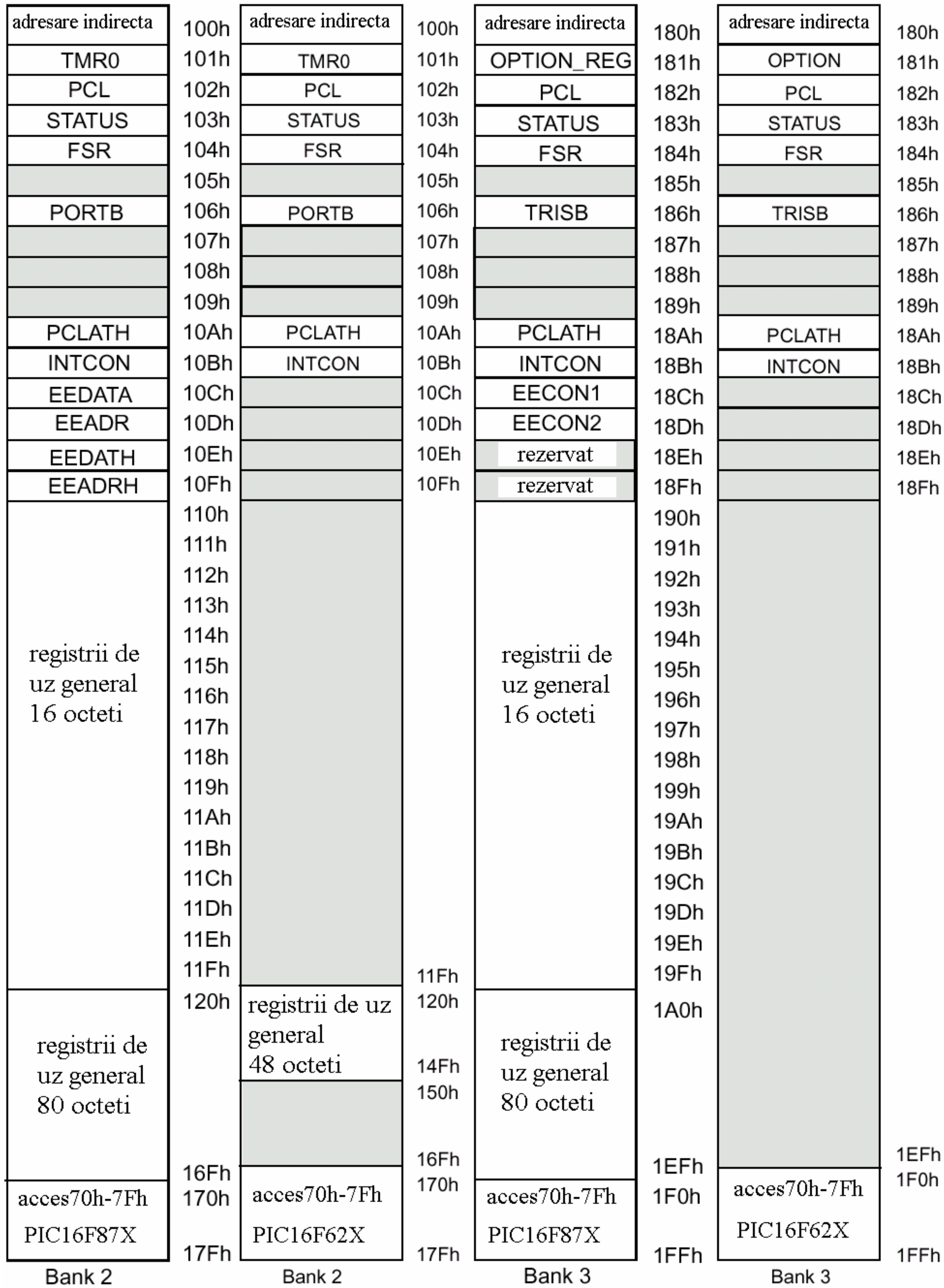


Fig.1-16 Regiştrii PIC în bank2 şi bank3

1.5.5 Oscilatorul, motorul microcontrolerului

La fel ca orice maşină, microcontrolerul necesită un motor pentru a putea funcţiona. Acesta este oscilatorul care generează tactul de procesor. Fără existenţa acestui tact nu se întâmplă nimic în regiştrii interni. Deoarece oscilatorul este specific fiecărui microcontroler, vom analiza tipurile de oscilatoare cu care poate funcţiona microcontrolerului PIC16F630, însă există mari asemănări cu situaţia descrisă şi în cazul celorlalte microcontrolere PIC:

- Oscilator extern cu cuarţ de frecvenţă medie (4MHz) sau rezonator ceramic în mod XT
- Oscilator extern de mică putere (32768Hz) în mod LP
- Oscilator extern sau rezonator ceramic de frecvenţă ridicată 10…20MHz în mod HS
- Oscilator extern RC cu două moduri de funcţionare în mod RC
- Oscilator intern RC cu două moduri de funcţionare în mod INTOSC
- Oscilator extern independent în mod EC

Modurile de funcţionare ale oscilatorului sunt setate în registrul *Configuration Word* prin cei mai puţin semnificativi biţi FOSC2…FOSC0 şi este evident că trebuie să existe o corelaţie între aceştia şi tipul de oscilator existent în mod real pe placa PCB:

FOSC2:FOSC0	Tipul de oscilator setat
111	RC, RA4 este CLKOUT, grupul RC se conectează pe RA5
110	RC, RA4 este I/O, grupul RC se conectează pe RA5
101	INTOSC, RA4 este CLKOUT, RA5 este pin I/O
100	INTOSC, RA4 este I/O, RA5 este I/O
011	EC, RA4 este I/O, RA5 este CLKIN
010	HS, cuarţul/rezonatorul se conectează între pinii RA4 şi RA5
001	XT, cuarţul/rezonatorul se conectează între pinii RA4 şi RA5
000	LP, cuarţul se conectează între pinii RA4 şi RA5

Configuraţiile hardware posibile sunt cele din figurile următoare:

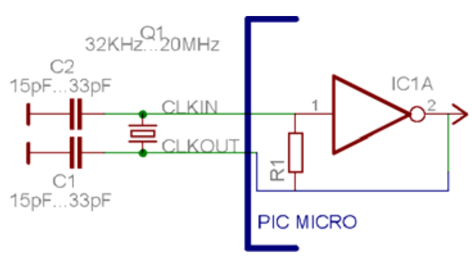


Fig.1-17 Configuraţia XT, HS (**H**igh **S**peed), LP (**L**ow **P**ower) cu cuarţ extern necesită condensatori de 15…33pF conectaţi la masă şi o conexiune cât mai scurtă a ansamblului până la capsula microcontrolerului. Componenta activă a oscilatorului este un inversor în PIC.

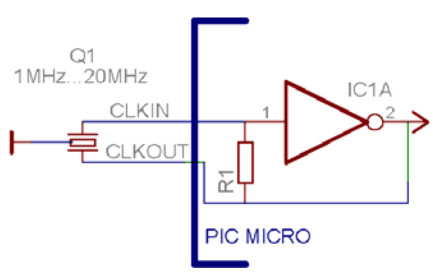


Fig.1-18 Configuraţia XT sau HS cu rezonator ceramic cu trei terminale nu mai necesită condensatori, aceştia sunt conectaţi intern în capsula rezonatorului Q1. Rezonatorul ceramic este mai puţin precis decât cuarţul şi mult mai instabil la variaţiile temperaturii ambiante.

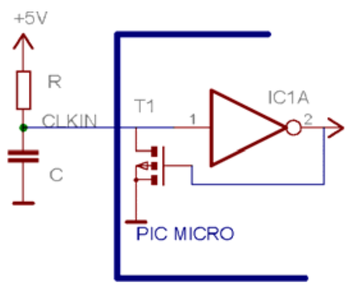


Fig.1-19 Configurația RC conectată pe CLKIN dă frecvența de oscilație, pinul CLKOUT poate fi pin de intrare/ieşire de uz general sau ieşire de control a frecvenței de oscilație divizată cu 4 (ieşire).

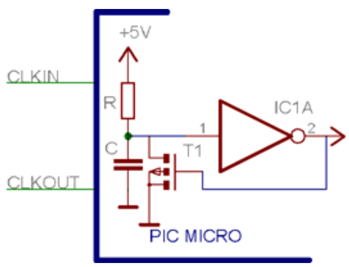


Fig.1-20 Configurația INTOSC (**INT**ernal **OSC**ilator) se bazează pe un condensator şi o rezistență internă a PIC-ului. CLKOUT poate fi pin de intrare/ieşire sau pin de control al frecvenței oscilatorului intern, divizată cu 4 (ieşire). CLKIN este în acest caz pin de intrare/ieşire de uz general. Unele microcontrolere dispun şi de un registru de calibrare OSCCAL pentru setarea frecvenței acestui oscilator.

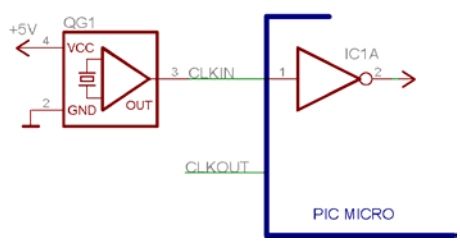


Fig.1-21 Configurația EC utilizează un oscilator independent extern a cărui ieşire de tact se conectează pe pinul CLKIN. Oscilatorul poate fi realizat şi cu porți NOT (NU). CLKOUT devine pin de intrare/ieşire de uz general. Mai multe microcontrolere pot fi alimentate cu tact de la acelaşi oscilator extern.

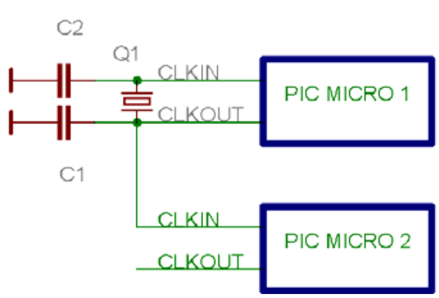


Fig.1-22 Conectarea a două microcontrolere PIC utilizând un singur oscilator extern. PIC MICRO1 este configurat în modul XT sau HS iar PIC MICRO2 este configurat în modul EC (**E**xternal **C**lock in)

Este important de ştiut că anumite configurații de setare a oscilatorului sunt mai energofage decât altele. De exemplu, curentul consumat în modul HS este mai mare decât în modul XT. Modul LP este cel care consumă cel mai puțin. De asemenea, numai anumite configurații permit trecerea microcontrolerului în modul SLEEP (adormit). Condensatoarele utilizate în modurile XT, HS şi LP au valoarea cuprinsă între 15pF şi 100pF, mai mari cu cât frecvența este mai mică. Creşterea capacității duce la creşterea stabilității oscilatorului dar şi la mărirea timpului de demarare al oscilatorului (start-up). Pentru aplicații ce necesită o frecvență extrem de precisă, C1 poate fi un trimer (condensator variabil) cu bună stabilitate termică.

1.5.6 Gata de start ? … nu fără setul de instrucțiuni !

O veste bună pentru cititorul probabil sufocat de atâtea informații greu de digerat, este că toate aceste microcontrolere utilizează acelaşi set restrâns de instrucțiuni de care aminteam. Există două mnemonice de bază (pseudolimbaj înțeles de asambloare) mnemonica standard a producătorului utilizată de majoritatea compilatoarelor sau asambloarelor (inclusiv JAL) şi mnemonica redusă utilizată de MPLAB (şi de JAL), care se referă la instrucțiuni simplificate ale celor dintâi dar care nu utilizează decât doi biți.

Familia microcontrolerelor Microchip cu performanțe medii de 8 biți, utilizează setul de instrucțiuni cu dimensiunea de 14 biți, alcătuit din 36 de instrucțiuni. Majoritatea acestora operează cu regiştrii f şi cu registrul special W care poartă numele de acumulator. Rezultatul operațiilor poate fi direcționat fie spre registru f, fie spre acumulator, fie spre ambii regiştrii în cazul anumitor instrucțiuni. Câteva instrucțiuni operează singure într-un registru de lucru f (de exemplu BSF). Deoarece traducerea mnemonicilor nu face decât să-l încurce pe utilizator, este păstrată descrierea şi funcția îndeplinită de instrucțiune în limba engleză. Instrucțiunile sunt grupate în trei categorii:

Operații literale şi de control

Hex	Mnemonică		Descriere	Funcția
3Ekk	ADDLW	k	Add literal to W	k + W->W
39kk	ANDLW	k	AND literal and W	k .AND. W->W
2kkk	CALL	k	Call subroutine	PC + 1->TOS, k->PC
0064	CLRWDT	T	Clear watchdog timer	0->WDT (şi prescaler-ul)
2kkk	GOTO	k	Goto address (k este 9 biți)	k->PC(9 biți)
38kk	IORLW	k	Incl. OR literal and W	k .OR. W->W
30kk	MOVLW	k	Move Literal to W	k->W
0062	OPTION		Load OPTION register	W->OPTION Register
0009	RETFIE		Return from Interrupt	TOS->PC, 1->GIE
34kk	RETLW	k	Return with literal in W	k->W, TOS->PC
0008	RETURN		Return from subroutine	TOS->PC
0063	SLEEP		Go into Standby Mode	0->WDT, stop oscillator
3Ckk	SUBLW	k	Subtract W from literal	k - W->W
006f	TRIS	f	Tristate port f	W->I/O control reg f
3Akk	XORLW	k	Exclusive OR literal and W	k .XOR. W->W

Operații cu octeți în regiştrii de lucru

Hex	Mnemonică		Descriere	Funcția
07ff	ADDWF	f,d	Add W and f	W + f-> d
05ff	ANDWF	f,d	AND W and f	W .AND. f-> d
018f	CLRF	f	Clear f	0->f
0100	CLRW		Clear W	0->W

09ff	COMF	f,d	Complement f	NOT. f->d
03ff	DECF	f,d	Decrement f	f - 1->d
0Bff	DECFSZ	f,d	Decrement f, skip if zero	f - 1->d, skip if 0
0Aff	INCF	f,d	Increment f	f + 1->d
0Fff	INCFSZ	f,d	Increment f, skip if zero	f + 1->d, skip if 0
04ff	IORWF	f,d	Inclusive OR W and f	W .OR. f->d
08ff	MOVF	f,d	Move f	f->d
008f	MOVWF	f	Move W to f	W->f
0000	NOP		No operation	
0Dff	RLF	f,d	Rotate left f	f(n)->dest(n+1), f(7)->C, C->dest(0)
0Cff	RRF	f,d	Rotate right f	f(n)->dest(n-1), f(0)->C,C->dest(7)
02ff	SUBWF	f,d	Subtract W from f	f - W->d
0Eff	SWAPF	f,d	Swap halves f	f(0:3)<->f(4:7)->d
06ff	XORWF	f,d	Exclusive OR W and f	W .XOR. f->d

Operații cu biți în regiştrii de lucru

Hex	Mnemonică		Descriere	Funcția
1bff	BCF	f,b	Bit clear f	0->f(b)
1bff	BSF	f,b	Bit set f	1->f(b)
1bff	BTFSC	f,b	Bit test, skip if clear	skip if f(b) = 0
1bff	BTFSS	f,b	Bit test, skip if set	skip if f(b) = 1

Pentru simplificarea scrierii unor instrucțiuni organizate la nivel de bit, ce au o constantă ca argument, s-au imaginat instrucțiuni ajutătoare recunoscute de MPLAB (mediul IDE al producătorului Microchip), numite instrucțiuni speciale sau *opcodes*. Aceste instrucțiuni sunt recunoscute şi de compilatorul JAL şi sunt în esență o succesiune de instrucțiuni standard formate fie numai din instrucțiuni ce operează cu biți, fie din instrucțiuni ce operează cu biți urmate de instrucțiuni de salt necondiționat, utilizate pentru operațiuni aritmetice sau logice:

Instrucțiuni speciale pe 2 biți (opcodes)

Mnemonică	Descriere	Operații Echivalente	Flagul afectat
ADDCF f,d	Add Carry to File	BTFSC 3,0	
		INCF f,d	Z
ADDDCF f,d	Add Digit Carry to File	BTFSC 3,1	
		INCF f,d	Z
B k	Branch	GOTO k	-
BC k	Branch on Carry	BTFSC 3,0	
		GOTO k	-
BDC k	Branch on Digit Carry	BTFSC 3,1	

		GOTO k	-
BNC k	Branch on No Carry	BTFSS 3,0	
		GOTO k	-
BNDC k	Branch on No Digit Carry	BTFSS 3,1	
		GOTO k	-
BNZ k	Branch on No Zero	BTFSS 3,2	
		GOTO k	-
BZ k	Branch on Zero	BTFSC 3,2	
		GOTO k	-
CLRC	Clear Carry	BCF 3,0	-
CLRDC	Clear Digit Carry	BCF 3,1	-
CLRZ	Clear Zero	BCF 3,2	-
LCALL k			
LGOTO k			
MOVFW f	Move File to W	MOVF f,0	Z
NEGF f,d	Negate File	COMF f,1	
		INCF f,d	Z
SETC	Set Carry	BSF 3,0	-
SETDC	Set Digit Carry	BSF 3,1	-
SETZ	Set Zero	BSF 3,2	-
SKPC	Skip on Carry	BTFSS 3,0	-
SKPDC	Skip on Digit Carry	BTFSS 3,1	-
SKPNC	Skip on No Carry	BTFSC 3,0	-
SKPNDC	Skip on No Digit Carry	BTFSC 3,1	-
SKPNZ	Skip on Non Zero	BTFSC 3,2	-
SKPZ	Skip on Zero	BTFSS 3,2	-
SUBCF f,d	Subtract Carry from File	BTFSC 3,0	
		DECF f,d	Z
SUBDCF f,d	Subtract Digit Carry from File	BTFSC 3,1	
		DECF f,d	Z
TSTFf	Test File	MOVF f,1	Z

Descrierea pe larg a setului de instrucțiuni este făcută în documentația fiecărui microcontroler în capitolul: "Instruction Set Summary, Instruction Description" însă tabelul prezentat mai sus are valoare de referință pentru utilizator.

Utilizarea setului de instrucțiuni va fi inevitabil în situația când trebuiesc tratate fenomene cu durate scurte de ordinul sutelor de microsecunde prin intermediul microcontrolerului. Duratele de timp lungi, de ordinul zecilor de milisecunde sau secunde pot fi tratate foarte bine utilizând limbajul JAL pur. Ce este acest limbaj şi care sunt particularitățile lui vom vedea în capitolul următor.

Bibliografie:

1. AN589 – A PC based development programmer for the PIC 16C84, DS000589A, Microchip Technology Inc. 1997
2. Interfacing the Standard Parallel Port , Interfacing the Extended Capabilities Port, Interfacing the Enhanced Parallel Port, http://www.beyondlogic.org/
3. Programator paralel, http://www.propic2.com
4. Eeprom Memory Program Specification PIC16C84, DS30189D, Microchip Technology, 1996
5. Eeprom Memory Program Specification PIC16FXXX, DS30925D, Microchip Technology, 1999
6. Software pentru programator serial sau paralel, http://www.ic-prog.com
7. Software universal pentru programator paralel, http://www.lpilsley.co.uk
8. Filă de catalog MAX232, Maxim Integrated Products, 1996
9. Interfacing the Serial/RS232 Port, V5.0, http://www.beyondlogic.org/
10. Editor profesional: Programmers File Editor, http://www.lancs.ac.uk/people/cpaap/pfe
11. Editor de amator, Jaledit, http://gum.sucks.nl/
12. IDE de amator, Jal Command Center, http://pic.flappie.nl
13. JAL compiler, http://www.voti.nl/jal/
14. Wloader, bootloader pentru PIC16F8xx, http://www.voti.nl/wloader/index_1.html
15. Wisp, Utilitar software pentru bootloader, http://www.voti.nl/wisp/index.html
16. MPLAB, http://www.microchip.com
17. File de catalog: PIC16C84, PIC16F62x, PIC16F7x, PIC16F87x, PIC12Fxxx, http://www.microchip.com
18. Program de învățare al arhitecturii PIC16F84, http://www.bubblesoftonline.com
19. Programator serial, http://www.jdm.homepage.dk
20. Programator USB, http://www.pic.flappie.nl



2 Ce este limbajul JAL ?

Jal (**J**ust **A**nother **L**anguage) este un limbaj de nivel înalt destinat tuturor microcontrolerelor flash din seria PIC12/16F: PIC16F8X, PIC16F62X, PIC16F87X, PIC16F7X, PIC16F676/630, 12C(F)50X, PIC12FXXX, parţial PIC18FXXX cât şi microcontrolerelor Scenix SX18 şi SX28. Jal este o alternativă la C sau PICbasic fiind însă un limbaj structurat care se potriveşte arhitecturii PIC-urilor. Seamănă forte mult cu limbajul Pascal, dar poate fi numit "Basic structurat" sau "ADA pentru microcontrolere". Majoritatea aspectelor limbajului sunt familiare oricui care are puţină experienţă în utilizarea a cel putin unui limbaj de nivel înalt. Câteva facilităţi mai exotice sunt pseudo variabilele, inexistenţa diferenţelor semantice dintre declaraţii şi instrucţiuni, denumirea implicită a parametrilor.

Compilatorul este disponibil în mod freeware [1], inclusiv codul sursă, (CD:\tools\jal_compiler), utilizatorul este liber să-l copieze şi să-l folosească pentru orice scop doreşte, însă autorul compilatorului, Wouter van Ooijen, trebuie anunţat prin email de orice utilizare semnificativă în vre-un proiect. Nu este oferită nici o garanţie pentru software-ul inclus, orice funcţionare defectuoasă sau distrugere a microcontrolerului cade în seama utilizatorului. Utilizarea compilatorului în aplicaţii medicale sau militare nu este recomandată chiar dacă este posibilă. Puteţi chiar să vindeţi produsele realizate cu acest compilator dar este interzis să ştergeţi sau să modificaţi nota privitoare la distribuţia sub licenţă GPL. Această obligaţie nu se referă la programul compilat care poate fi vândut sau cedat împreună cu un produs care conţine bibliotecile în forma compilată.

Toate reacţiile utilizatorilor (experienţe, sugestii, proiecte, defecţiuni) sunt binevenite. Ele pot fi aduse la cunoştinţa oricărui autor al acestei cărţi, în limba română (vsurducan@gmail.com) sau engleză (wouter@voti.nl). Orice bibliotecă nouă, creată de dumnevoastră poate fi integrată în noua versiune.

2.1 Limbajul

2.1.1 Noţiuni de bază

2.1.1.1 Formatul

Limbajul Jal are un format liber (excepţie fac comentariile care trebuiesc precedate de simbolurile -- sau ;) şi nu este sensibil la majuscule sau minuscule cu excepţia variabilelor şi a numelui filelor incluse. Toate caracterele având o valoare ASCII mai mică decât

"*space*" (tab, carriage return, linie nouă, form feed, etc) sunt tratate ca spaţii, cu excepţia când o linie se termină cu un comentariu. Jal nu foloseste nici un separator în corpul instrucţiunii, singurul separator real este virgula dintre argumentele actuale (curente) sau formale în cadrul unei proceduri sau funcţii. Jal nu are etichetă. Sintaxa limbajului este bazată pe separatori, este obligatoriu utilizarea unui spaţiu între diverşi identificatori, operatori, etc.

```
   function f ( byte in a ) ; a este parametru formal
   X = f ( 2 )              ; 2 este parametru actual
   -- instrucţiunea if…then…else…end if :
   if a > b then  a = b + 1
   else           a = b - 1
   end if
         -- dar acesta instrucţiune are acelalaşi efect:
    if a  b then a = b + 1 else a = b - 1 end if
         -- virgulele sunt necesare între argumentele actuale
f(a, b, c, d)
```

2.1.1.2 Comentarii

Un comentariu trebuie precedat de unul din următoarele caractere: -- sau ; şi continuă până la sfârşitul liniei în curs.

```
   -- linia urmatoare conţine un comentariu după incrementare
   ticks = ticks + 1 -- înca un tick
   -- linia următoare conţine o eroare: *--
   b = 2 *-- acesta nu va fi interpretat ca şi un comentariu valid
   ; linia următoare este identică cu prima
   ticks = ticks + 1 ; acesta este un comentariu valid
```

2.1.1.3 File incluse

O incluziune are ca efect citirea fişierului a cărui nume urmează după directiva "include". Dacă dintr-o eroare coexistă mai multe incluziuni având aceleaşi nume, este păstrată doar prima. Este bine ca utilizatorul să verifice diversele biblioteci scrise de el înainte de compilare pentru a nu avea totuşi surprize. O bibliotecă poate include toate bibliotecile de nivel mai scăzut necesare, (situaţia se regăseşte de fiecare dată când se apelează biblioteci ce conţin informaţii privitoare la structura hardware şi anume conectarea pinilor sau iniţializări ale diverselor module electronice).
Filele incluse sunt căutate întâi în directoarea curentă iar apoi în fiecare locaţie indicată de căutatorul de directoare al compilatorului. Această particularitate permite ca o bibliotecă standard să fie înlocuită cu altă bibliotecă specifică. Acest lucru obligă utilizatorul să menţioneze directoarele proiectelor în calea în care compilatorul caută (search path). Incluziunile pot fi imbricate la orice nivel (în programul principal sau în orice biblioteci adiacente).

```
   include jlib   -- include biblioteca jal standard
   include i2c    -- include biblioteca i2c
```

2.1.1.4 Programul

Un program jal este o înşiruire de instrucţiuni. Declaraţiile sunt considerate de asemenea instrucţiuni, deci pot apare aproape oriunde în program. Este nevoie totuşi ca declaraţiile să apară înaintea instrucţiunilor care operează cu ele.

```
procedure p is
   var byte a -- declaraţie la inceput de bloc
   a = 5
   var byte b -- declaraţie între două instrucţiuni
   b = a
end procedure
```

2.1.1.5 Declaraţii

Jal este un limbaj structurat la nivel de blocuri, fiecare declaraţie este vizibilă din momentul declarării şi până la sfârşitul blocului în care declaraţia apare (până la primul `end` al nivelului curent). O declaraţie poate ascunde o declaraţie cu acelaşi nume dintr-un bloc ce a fost închis (o procedură sau o funcţie). Declararea unor regiştrii implicaţi într-un bloc scris în limbaj de asamblare poate fi făcută înafara blocului respectiv, ca instrucţiune de tip jal.

```
procedure read_config (byte out config_status ) is
   -- corpul procedurii, config_status este parametru de
   -- iesire pentru procedura in curs
end procedure

var byte config_status -- variabila trebuie redefinită
read_config ( config_status )
           -- şi procedura poate fi apelată, sau:
var byte nume_dorit
read_config ( nume_dorit )
           -- variantă funcţional identica cu cea anterioară
var byte counter   -- definirea octetului de lucru
procedure example_assembler is
           -- o procedură în limbaj de asamblare
 assembler -- care începe aici
  local loop1
           -- conţine etichete locale, valabile doar…
  loop1:  incf  counter
           -- în interiorul blocului assembler…
       btfsc counter, 7
       goto loop1
       return
 end assembler –- sfârşitul blocului în assembler
end procedure
```

O declaraţie nu poate ascunde un nume care a fost deja declarat la acelaşi nivel, este nevoie de o nouă declaraţie cu un nume schimbat.

```
var byte b       ; se defineşte octetul b
```

```
  while b > 0 loop
     var bit b
-- rescrie octetul b, acesta nu mai are valoarea inițială, fiind
-- acum bitul b
     b = false    -- se referă la bitul b şi nu la octetul inițial
     var byte b
-- aceasta este o eroare; octetul b a fost deja definit !
  end loop
```

2.2 Tipuri specifice

2.2.1 Bit

Bitul este unitatea de bază a sistemului binar, ea are doar două valori: **on** (**true**, **high**) şi **off** (**false**, **low**). Instrucțiunile ***if…then…else…end if*** şi ***while…loop…end loop*** operează numai cu biți.

```
  var bit a = high
  var byte x
  a = x + 5 -- eroare ! bitul a nu poate lua valoarea unui octet
```

2.2.2 Byte sau octet

Un byte este un întreg format din 8 biți reprezentat ca modulo 256. Valorile negative sunt înterpretate ca modulo 256 deci –1 şi 255 sunt două notații care înseamnă acelaşi lucru. Utilizatorul trebuie deci să țină cont că cel mai mare număr pozitiv cu care poate lucra pe un octet este 255 (0x_FFh , 0b_1111_1111)

```
   var byte n = 1, m = 257 ; 257 este ințeles ca 2 (257-255 = 2)
  if n == m then ; instrucțiunea prezentă aici va fi executată dacă
                 ; condiția este indeplinită
  end if
```

2.2.3 Universal

Tipul universal există doar la momentul compilării. O expresie care nu este forțată să fie de un tip anume este implicit universală. Acest tip de expresie poate implica doar constante, literale şi operatori interni care sunt evaluați de compilator ca întregi cu semn, de maxim 32 de biți.

```
 const xtal = 10_000_000 ; frecvența cuațului este de 10 MHz
 const mips = xtal / 4   ; durata celui mai mic tact intern
```

2.3 Formate numerice

2.3.1 Bit

Bitul are două valori **on** (**true**, **high**) reprezentând "1" logic şi **off** (**false**, **low**) reprezentând "0" logic.

```
pin_a0 = low  ; se alocă pinului a0 valoarea logică 0
pin_b0 = high ; se alocă pinului b0 valoarea logică 1
```

2.3.2 Universal

Valorile numerice pot fi scrise în baza de numeraţie 2, baza 10 sau baza 16. Implicit ele apar în baza 10. O altă bază de numeraţie este specificată prin prefixul 0b pentru baza 2 , 0d pentru baza 10 şi 0x sau 0h pentru baza 16. Un format numeric este de tip universal. O expresie de tip universal poate fi utilizată ori de câte ori este necesar un octet, deci formatele numerice pot fi utilizate ca şi octeţi. Barele de separaţie la nivel de 4 biţi (*nibble*) sunt ignorate, ele au doar rolul de a ordona întregul număr.

```
0b_0101_0101 -- reprezentarea binară a unui octet
0x_55        -- aceeaşi valoare reprezentată hexazecimal
85    -- aceeaşi valoare reprezentată zecimal
```

2.3.3 ASCII

Un cuvânt ASCII este o notaţie alternativă pentru valoarea codului ASCII al caracterului indicat. Un cuvânt ASCII poate conţine un singur caracter şi aparţine tipului universal.

```
"A" -- A majuscul
"a" -- a minuscul
```

2.3.4 Constante

Prin declararea unei constante se defineşte un nume care are o valoare constantă. Când tipul constantei este omis, aceasta este de tip universal. Printr-o singură declarare a unei constante se pot introduce un numar mare de constante de acelaşi tip. Constanta se utilizează doar la momentul compilării şi poate fi de maxim 32 biţi, de tip întreg.

```
const byte cr = 0x0D, lf = 0x0A
-- constante de tip octet
const seconds_per_day = 60 * 60 * 24
-- constante universale de tip întreg
const baud_rate = 115200
```

2.3.5 Variabile

2.3.5.1 Declararea unei variabile

O variabilă declarată defineşte un nume căruia îi corespunde o locație de memorie (registru sau port al microcontrolerului).

```
   var volatile byte pir1    at 0x0C
-- octetul are adresa fizică 0x0C ! şi este volatil
-- adică compilatorul îi va aloca aceeaşi adresă de fiecare dată
-- indiferent de numarul de compilări sau de lungimea codului
```

Opțional, numele poate reprezenta o locație specifică, dacă nu e nevoie de acest lucru compilatorul îi alocă un registru oarecare. Alocarea nu este reproductibilă pentru diverse compilări succesive dacă locația specifică nu este menționată.

```
     var byte pir1
-- octetul are adresa pe care compilatorul o alocă automat şi nu adresa pe care utilizatorul o doreşte
```

Opțional, o valoare poate fi asignată unei variabile, acest lucru are acelaşi efect ca şi o asignare echivalentă urmată imediat unei declarații. Valoarea inițială nu este nevoie să fie o expresie constantă.

```
     var byte hour = 12 -- acest lucru este identic cu
     var byte hour      -- declararea variabilei şi
     hour = 12          -- asignarea unei valori
```

O singură declarație a unei variabile poate introduce un număr mai mare de variabile care trebuie să fie toate de acelaşi tip.

```
     var byte x, y = 3, z = f( 14 )

-- diverse reprezentari de variabile de tip octet
```

2.3.5.2 Poziția unei variabile

Declararea unei variabile poate specifica direct sau indirect adresa variabilei. Adresa este interpretată ca un registru de tip octet iar pentru variabilele de tip bit, de poziția bitului respectiv în cadrul octetului, 0 fiind bitul cel mai puțin semnificativ.

```
   var volatile byte port_a at 0x06
   -- portul A de intrare ieşire al PIC-ului
   var volatile bit status_z at 3 : 2
   -- flagul de zero, bitul 2 al registrului status în bank_0
```

Toate adresele expresiilor trebuie să fie la momentul compilării de tip constat (ele trebuie să fie definite în prealabil într-o bibliotecă). Numele unei variabile poate fi folosit ca o adresă pentru un octet, interpretat ca şi octetul de adresă al variabilei.

```
   var byte adresa    -- declararea unui octet
   var byte octet_curent at adresa
             -- octet_curent este definit la "adresa"
   var bit bit_zero at octet_curent : 0
             -- bitul zero al octet_curent
```

Adresarea variabilelor ignoră organizarea pe bancuri de memorie pentru unele microcontrolere ce sunt suportate complet de compilator (PIC16F84). Fiecare adresă indică un registru adresabil diferit. Compilatorul ține cont de translația necesară în bancul de memorie în care se găseşte registrul. Jal 04.6x generează cod hexa pentru întreaga memorie a PIC-urilor acceptate (8 K), dar nu alocă variabile decât în pagina sau bancul 0.

2.3.5.3 Variabila volatilă

O variabilă poate fi declarată volatilă, acest lucru înseamna că variabila nu posedă forma semantică normală (este de obicei utilizată pentru definirea regiştrilor cu funcții speciale). Pentru **variabilele nevolatile** compilatorul ia în considerare forma semantică normală a acestora:

- Asignarea (alocarea) poate fi optimizată de catre compilator dacă acelaşi efect poate fi obținut printr-o altă metodă (ca de exemplu prin substituirea valorii asignate când variabila este considerată referință)
- Variabila va conține întotdeauna ultima valoare asignată.

Pentru o **variabilă volatilă:**

- Toate asignarile variabilei vor fi făcute exact cum s-a specificat de către utilizator.
- Nu se aşteaptă ca variabilele să conțină ultima valoare asignată.

Pentru variabile nevolatile, compilatorul poate optimiza "după propria dorință" atât timp cât efectul observabil asupra variabilelor volatile rămâne identic. Aceasta poate include chiar ştergerea unor asignări care nu sunt necesare.

```
   var volatile byte FSR    at 4
-- registru de adresare indirectă FSR
   var volatile byte INDF   at 0
-- registru de date pentru adresare indirectă
   var volatile byte count
-- acesta este un registru numărător
```

2.3.5.4 Înlocuitori

O variabilă poate fi declarată pentru a fi un înlocuitor pentru o altă variabilă. Această facilitate este utilizată mai mult ca o declarație de constantă, pentru a ascunde actuala identitate al unui identificator, într-o porțiune de cod. O variabilă înlocuită conține adresa variabilei pe care o înlocuieşte şi nu conținutul variabilei volatile respective.

```
-- un fragment al bibliotecii i2cp care defineşte pinii utilizați:
   var byte volatile i2c_clock    is pin_a3
   var byte volatile i2c_data_in  is pin_a4
   var byte volatile i2c_data_out is pin_a4_direction
```

2.4 Expresii matematice

2.4.1 Elemente

O expresie este alcătuită din numere, identificatori, funcţii sau proceduri şi operatori. Un identificator poate identifica o constantă, o variabilă sau un parametru formal existent într-un subprogram.

2.4.2 Operatori matematici

Operatorii predefiniți sunt următorii, prioritatea maximă fiind 5:

operator	prioritate	interpretare	argument în stânga	argument în dreapta	rezultat
!	5	nu (element)		bit	bit
!	5	nu (element)		byte	byte
+	5	plus (element)		byte	byte
-	5	minus (element)		byte	byte
*	4	înmultire	byte	byte	byte
/	4	împărtire	byte	byte	byte
%	4	modulo	byte	byte	byte
+	3	plus	byte	byte	byte
-	3	minus	byte	byte	byte
<<	2	rotire la stânga	byte	byte	byte
>>	2	rotire la dreapta	byte	byte	byte
>	2	mai mare dacât	byte	byte	bit
<	2	mai mic decât	byte	byte	bit
>=	2	mai mare sau egal	byte	byte	bit
< =	2	mai mic sau egal	byte	byte	bit
==	2	egal	byte	byte	bit
!=	2	diferit	byte	byte	bit
&	1	şi	bit	bit	bit
&	1	şi	byte	byte	byte
|	1	sau	bit	bit	bit
|	1	sau	byte	byte	byte
^	1	sau exclusiv	bit	bit	bit
^	1	sau exclusiv	byte	byte	byte

Operatorii predefiniți nu pot fi redeclarați, aceşti operatori au un scop precis fixat. Alți operatori matematici pot fi declarați sau redeclarați de utilizator. Toți operatorii care lucrează cu octeți pot de asemenea să opereze cu tipuri universale. Când rezultatul operării

cu un argument de tip octet este un bit, rezultatul operării cu un argument universal va fi tot de tip bit. Operarea cu cuvinte de 16, 24 şi 32 de biţi este deasemenea posibilă la nivelul procedurilor scrise în assembler sau JAL.

2.4.3 Priorităţi

Parantezele pot fi utilizate pentru a forţa diverse asocieri, altfel priorităţile sunt conforme tabelului anterior. Pentru operatori matematici cu aceeaşi prioritate, operatorul aflat în stânga are prioritatea maximă.

```
var byte x = ! a + b
 -- (! a) + b prioritatea maximă o are negarea
var y = ! ( a + b )
 -- () paranteze utilizate pentru a forţa o altă  interpretare a
 -- aceleiaşi expresii
```

2.4.4 Ordinea evaluării

Ordinea în care diverse părţi ale unei expresii este evaluată, nu este definită. O parte a expresiei poate fi evaluată de mai multe ori, o altă parte care nu are influenţă asupra valorii finale a expresiei poate să nu fie evaluată de loc.

```
var byte n = 1
function f return byte is
   n = n + 1
   return 3
end function

function g return byte is
   n = 2 * n
   return 4
end function

var byte a = f + g
if n == 4 then
   -- linia utilizatorului
end if
```

2.5 Instrucţiuni

2.5.1 Declaraţii

Declaraţiile sunt considerate instrucţiuni, deci pot apare oriunde într-un program unde instrucţiunile sunt admise.

```
   a = f( 9 )
   var byte x = 1, y = 0
-- avem nevoie de câteva declaraţii locale, nici o problemă
   while x < a loop
      y = y + x
      x = x + 1     end loop
```

2.5.2 Asignări

O instrucțiune de asignare evaluează expresia şi îi înlocuieşte valoarea cu variabila sau parametrul formal indicat de numele din stânga numărului asignat.

```
   var byte a
   procedure p( byte out q ) is
      q = 5       -- parametrul de ieşire q ia valoarea 5
      a = 4       -- variabila globală a ia valoarea 5
   end procedure
         a = 5
-- noua variabilă locală a ia valoarea 5
```

2.5.3 If

O instrucțiune ***if*** evaluează întreaga expresie ce urmează. Dacă rezultatul este adevărat, este executată întreaga listă de instrucțiuni ce urmeaza după ***if***. Forma generală a instrucțiunii este*:* ***if…then…elsif…else…end if.*** Înaintea lui ***else*** este permis orice număr de ***elsif***. Când condiția ***if*** este falsă, este evaluată prima condiție ***elsif***. Dacă aceasta este adevărată, instrucțiunile corespunzătoare sunt evaluate, dacă nu, execuția continuă cu următorul ***elsif.*** Când nici una din condițiile ***if*** sau ***elsif*** nu este adevărată, sunt executate instrucțiunile din partea opțională ***else***. Toate expresiile din ***if…end if*** trebuie să fie de tip bit.

<table>
<tr>
<td><pre>if a < b then
   x = a
else x = b
end if</pre></td>
<td><pre>x = x + 1
if x > 10 then
   x = x - 10
end if</pre></td>
</tr>
</table>

```
   if    target_clock == 10_000_000 then
      -- codul pentru oscilator de 10MHz
   elsif target_clock == 4_000_000 then
      -- codul pentru oscilator de 4MHz
   elsif target_clock == 32_768
      -- codul pentru oscilator de 32.768kHz
   else   -- ce ne facem acum ?
      pragama error -- oscilator necunoscut
   end if
```

2.5.4 While

O instrucțiune ***while…loop…end loop*** evaluează expresia ce urmează după ***while***. Dacă rezultatul este fals, întreaga instrucțiune îşi termină execuția. Dacă rezultatul este adevărat, sunt executate instrucțiunile ce urmează, după care expresiile sunt evaluate din nou. Toate expresiile din cadrul buclei ***while*** trebuie să fie de tip bit.

```
   procedure div_rem (  bit in x, bit in y
                       bit out d, bit out r ) is
      if y == 0 then -- ce să facem aici ?
      else
         r = x
         d = 0
         while r > y loop
```

```
         d = d + 1
         r = r - y
      end loop
   end if
end procedure
```

Exista posibilitatea ca o buclă de tip ***while*** sa fie intreruptă de utilizator cu o condiție impusă:

```
   var bit test_bit = high
   while test_bit loop
-- aici pot interveni şi alte instrucţiuni
     counter = counter + 1
-- counterul va fi incrementat doar odată
     test_bit = low ; după care se iese forţat din buclă
   end loop
```

2.5.5 For

O instrucțiune ***for*** determină ca expresiile ce urmează să fie executate de numărul de ori indicat.

```
   procedure delay_1S is
-- obţinerea unei întârzieri de 1 secundă
      for 100 loop
         for 100 loop
            delay_100uS
-- utilizând întârzieri de 100uS
         end loop
      end loop
   end procedure
```

2.5.6 Forever

Instrucțiunea ***forever loop…end loop*** determină ca toate expresiile din buclă să fie executate la nesfârşit. Are acelaşi efect ca şi instrucțiunea ***while…loop*** cu o condiție adevărată permanentă. Se utilizează de regulă la obținerea unui program ciclic. Este instrucțiunea ce dictează repetarea algoritmului definit la infinit şi poate genera programul principal (main loop):

```
    forever loop
      pin_a0 = true
      delay_1S
      pin_a0 = false
      delay_1S
   end loop
```

2.5.7 Definirea procedurilor

O procedură este definită de un nume urmat în paranteze de n argumente curente (active doar în cadrul procedurii). Parametrii care au direcții de intrare (in) sau de ieşire (out) iau valorile indicate de argumentul curent. Apoi expresiile care formează corpul

procedurii sunt executate şi argumentele curente care corespund parametrilor de intrare sau de ieşire iau valoarea acelui parametru.

```
   procedure read_config ( byte out config_status ) is
      reset = high
      out_3w ( 0x_ac )         -- procedura apelată
      in_3w  ( config_status )
-- altă procedură apelată, acelaşi parametru de ieşire care va fi
-- transferat procedurii definite (read_config)
      reset = low
   end procedure

   var byte config_status      -- variabila trebuie redefinită
   read_config ( config_status )
-- după care procedura se poate apela :
   var byte result
   read_config ( result )
-- aceasta este altă soluție de apelare a aceleiaşi proceduri
```

Pentru asocierea dintre argumentele curente şi parametrii, există următoarea regulă: când nu există nici un parametru curent corespunzător, este utilizat parametrul declarat în cadrul procedurii (care este implicit). Când nici parametrul curent nici cel declarat nu sunt utilizați, se generează o eroare. In corpul procedurii, asignarea unor parametrii de tip ieşire sau intrare-ieşire poate afecta imediat parametrul curent (pass by reference) sau acesta poate fi afectat numai la sfârşitul procedurii (pass by value-result). Compilatorul este liber să aleagă. O procedură care nu transferă parametrii nu are paranteze.

```
   var byte a
  procedure  p( byte in out x = a, byte in q = 5 ) is
      a = 0   x = q
      if a == q then  -- fă ceva !
      end if
  end procedure
   p( a, 11 )    -- doi parametri curenți
   p             -- identic cu p(a, 5)
   p( 12 )       -- eroare, 12 nu poate fi parametru curent pentru x
```

2.5.8 Return

O instrucțiune ***return*** este utilizată pentru a termina necondiționat execuția unei proceduri sau a unei funcții. Pentru o funcție, instrucțiunea ***return*** trebuie urmată de o expresie având tipul corespunzator (bit sau octet).

```
   function root( byte in x ) return byte is
      var byte n = 15
      forever loop
         if n * n <= x then
            return n
         end if
         n = n - 1
      end loop
    end function
```

2.5.9 Assembler

Instrucțiunea ***assembler*** simplă, este constituită din cuvântul ***asm*** urmat de o singură mnemonică a limbajului de asamblare (vezi instrucțiunile de asamblare ale microcontrolelor PIC). Instrucțiunea ***assembler*** completă, este constituită din cuvântul ***assembler***, o secvență de declarare de etichete, etichete şi mnemonici şi se termină cu secvența ***end assembler***. O etichetă trebuie declarată înainte de a fi utilizată iar o etichetă declarată trebuie utilizată o singură dată pentru a defini o locație de salt. Eticheta trebuie folosită de la linia în care este declarată şi până la sfârşitul blocului marcat cu ***end assembler***. Expresiile utilizate ca argumente în limbajul de asamblare, trebuie să fie la momentul compilării de tip constant. Variabilele utilizate în aceste expresii sunt evaluate conform adreselor lor existente în ***file register*** (vezi structura microcontrolerului PIC). Când este necesar, compilatorul va translata adresele variabile în diversele bancuri de memorie ale microcontrolerului. Utilizatorul este responsabil pentru a seta pagina de memorie şi bancul corespunzator utilizând mnemonicele ***page*** sau ***bank***. Pentru PIC16F84 mnemonicile ***page*** şi ***bank*** sunt ignorate.

```
asm clrwdt -- instrucțiune simplă
procedure first_set( byte in x, byte out n ) is
   assembler
-- urmează un bloc în limbaj de asamblare
      local loop, done  ; declararea de etichete de salt
      clrf n            ; şterge variabila n
   loop :               ; execută în buclă ce urmează
      btfsc x, 0        ; ieşi din buclă dacă bitul 0 al x este 1
         goto done
      incfsz n, f       ; dacă nu incrementează n
      rrf x             ; roteşte dreapta x
      goto loop         ; şi reia
   done :
   end assembler
end procedure
```

2.6 Subprograme

2.6.1 Proceduri

O procedură declarată este alcătuită dintr-un nume pentru o listă de argumente şi o secvență de instrucțiuni. Mecanismul de transfer al argumentelor este descris în secțiunea ***definirea procedurilor***.

```
procedure zero( byte out x, byte in y ) is
 if y > 0 then x = y end if
end procedure
```

2.6.2 Funcții

O funcție declarată este alcătuită dintr-un nume pentru o listă de argumente, o secvență de instrucțiuni şi o instrucțiune *return*. Când execuția unei instrucțiuni atinge sfârşitul înşiruirii de instrucțiuni, valoarea returnată prin comanda *return* este nedefinită. Tipul returnat poate fi octet sau bit. Se pot returna n valori având tipurile precizate anterior.

```
   function reverse( byte in x ) return byte is
      byte y
      for 8 loop
         asm rrf x, f
         asm rlf y, f
      end loop
      return y
   end function
-- aceasta funcţie inversează ordinea biţilor (lsb,msb)într-un octet
```

2.6.3 Pseudo-variable

O pseudo-variabilă este în esenţă o rutină de acces ce poate fi utilizată ca orice variabilă, ea este obligatoriu implementată într-o rutină de tip ***get*** (ia) şi/sau ***put*** (pune). Una dintre cele două tipuri de rutine poate fi omisă, astfel variabila poate fi ***read-only*** sau ***write-only.*** In mod alternativ pot fi declarate o variabilă şi o rutină de tip ***get*** sau ***put***, caz în care variabila simplă va fi utilizată în locul rutinei lipsă. O procedură ***put*** trebuie să aibă ca parametru un octet iar o funcţie ***get*** trebuie să nu aibă nici un parametru. Utilizarea pseudo-variabilei poate fi făcută într-o expresie, la stânga unei declaraţii sau ca un parametru actual. Scopul pseudo-variabilei este să ascundă o secvenţă complexă de program (de exemplu un protocol de comunicaţie sau de afişare pe LCD).

```
procedure hd44780'put( byte in x ) is ...
   hd44780 = "H"  hd44780 = "e" hd44780 = "l" hd44780 = "l"
  hd44780 = "o"
end procedure

   procedure async'put( byte in x ) is … end procedure
   function async'get return byte is … end procedure

   forever loop
      byte c = async
      if ( c = "a" ) & ( c <= "z" ) then
         c = c + "A" - "a"
      end if
      async = c
   end loop
```

2.7 Pragma's

2.7.1 Nume

Pragma name poate fi utilizată pentru o bună documentare a numelui filei sursă. Compilatorul verifică dacă numele în cauză este într-adevar numele real al fişierului. (extensia fişierului, *.jal trebuie omisă).

```
   -- un comentariu poate să nu însemne nimic:
   -- aceasta este fila xyz
   -- dar când urmatoare linie este compilată
   -- aceasta va fi într-adevar fila e0001!
   pragma name e0001
```

2.7.2 Specificarea tipului microcontrolerului (pragma target)

Pragma target dă compilatorului informații despre ținta (microcontrolerul) care va fi utilizată, *target chip* (**16F84**, **16F87x**, **SX18**, **SX28, etc.**) şi setările oscilatorului (**hs**, **xt**, **rc**, **lp** sau **internal**) care trebuie specificate. Opțional pot fi specificate: starea watchdog-ului (**on** sau **off**, implicit **off**), a protecției la citire (**on** or **off**, implicit **off**) şi a întârzierii la pornire (**on** sau **off**, implicit **on**). Frecvența oscilatorului de tact nu-i este de folos compilatorului, dar unele biblioteci (busy delay, interval delay, hd44780, asynch) au nevoie să cunoască frecvența ceasului care este dată de variabila globală pre-declarată, ***target_clock.*** Este posibil ca toate ***pragma***-urile să fie puse într-o filă sursă a proiectului, dar este mult mai uşor să fie incluse în una din bibliotecile standard ale microcontrolelor acceptate (16f84_4, SX28_50 etc.).

```
pragma target chip        16c84
pragma target clock       4_000_000
pragma target osc         xt
pragma target watchdog    off
pragma target powerup     on
pragma target protection  off
```

Pragma target fuses lasă la latitudinea utilizatorului configurarea tuturor fuzibilelor specifice pentru un microcontroler. Este utilă în situația când microcontrolerul are foarte multe opțiuni pentru oscilator (cazul lui PIC16F62x) sau configurația dorită de utilizator nu este prezentă în biblioteci.

```
pragma target fuses       0x_3fc1
pragma target fuses       0b_0011_1111_1100_0001

-- word  cpd lvp boden mclr pwrt wdt osc
-- ---------------------------------------------
-- 3fc1  off on  on    io   off  off xt
-- 3fd0  off on  on    io   off  off intrc+io
-- 3ff0  off on  on    mclr off  off intrc+io
-- 3f62  off off on    mclr off  off hs
```

2.7.3 Salt la o adresă de tabel (jump_table)

Pragma jump_table informează compilatorul că subprogramul curent conține cod maşină care va modifica registrul ***program counter*** utilizând registrul ***PCL***. Compilatorul se va asigura că biții registrului ***PCLATH*** sunt setați corespunzator (aceasta afectează pagina de memorie în care se lucrează) . O rutină care conține **pragma jump_table** este codată diferit în funcție de versiunea de compilator; în ultima pagină de memorie la variantele inițiale (până la jal.04-40) şi imediat după codul utilizator pentru ultimele versiuni. Tabelul de salt pentru variabila volatilă este pus deasemenea aici. Când tabelul şi rutinele nu încap în ultima pagină de memorie, este generat un mesaj de eroare. De asemenea când un tabel de salt sau o rutină conținând această **pragma** este prezentă, selectarea biților de pagină se face automat.

```
procedure _seven_table is
      pragma jump_table
         assembler
```

```
            addwf 2, f
            retlw seven_0
            ...
            retlw seven_f
      end assembler
   end procedure
```

2.7.4 Eroare

Pragma error produce o eroare de compilare când compilatorul ajunge la faza de generare a codului maşină. Poate fi utilizată pentru a testa diverse erori la momentul compilării, ca de exemplu o frecvenţă de ceas necorespunzătoare. Notaţi că evaluarea de către compilator a expresiilor constante şi a parantezelor inutile au loc chiar dacă *switch*-ul de optimizare nu este activ, următoarele expresii vor funcţiona chiar fără optimizare:

```
if target_clock < 4_000_000 then
      pragma error
-- frecvenţa de tact trebuie sa fie < 4 MHz
end if
```

2.7.5 Test

Pragma test poate fi utilizată pentru două scopuri: testarea mecanismului detector de erori al compilatorului şi testarea codului generat de compilator.
Pragma test catch arată că următoarea linie va cauza o eroare de compilare la linia indicată. Când aceasta este situaţia reală, compilatorul va returna un mesaj de succes, altfel va returna un mesaj de eşec.

```
   var byte n
   pragma test catch 9
   var bit n
```

Pragma test assert arată că după execuţia simulată în acest punct al programului, variabila indicată trebuie să aibă valoarea definită de utilizator. Compilatorul conţine un simulator care este activat de optiunea –t. ***Pragma test done*** arată că simularea în curs trebuie să se termine. Compilatorul va returna un mesaj de succes sau de eşec.

```
   var byte a, b, c
   a = 5
   b = 6
   c = a * b
   pragma test assert c == 30
   c = a % b
   pragma test assert c == 5
   pragma test done
```

2.7.6 Eedata

Pragma eedata scrie în memoria eeprom, datele (caracterele ASCII în exemplu) ce urmează după instrucţiune. Acestea trebuie să fie separate de virgulă şi după ultimul caracter trebuie să apară cifra 0. Această instrucţiune nu poate fi simulată în MPLAB,

verificarea corectitudinii ei se poate face doar importând şi citind fila *.hex generată de compilator.

```
pragma eedata "H", "e", "l", "l", "o", " ", 0
```

2.7.7 Keep page, bank

Pragma keep page, bank va menţine configuraţia codului scris în limbaj de asamblare ce urmează, indiferent de pagina sau bancul de memorie în care compilatorul îl va asambla, cu alte cuvinte rezultatul compilării se supune principiului WYSWYG ("what you see is what you get")

```
pragma keep page, bank
assembler
            local outer_loop, inner_loop
-- ia argumentul şi înlocuieşte-l cu 0 dacă e prea mic
                 movlw   yy
            bank addwf   x, w
                 skpc
                 movlw 0
            -- fă o copie locală
            bank movwf   outer_counter
                 incf    outer_counter, f
            -- bucla exterioară durează 1uS
         outer_loop:
            -- bucla interioară durează 6 + 4 * n
                 movlw zz
            bank movwf   inner_counter
            page inner_loop
         inner_loop:
                 decfsz  inner_counter, f
                 goto    inner_loop
            -- bucla exterioară din nou?
            page outer_loop
            bank decfsz  outer_counter, f
                 goto    outer_loop
end assembler
```

2.7.8 Interrupt

Generarea codului în cazul în care nu se utilizează întreruperi, începe de la adresa 0. Când una sau mai multe întreruperi sunt prezente, codul începe cu un salt la rutina de întreruperi (adresa 4), iar rutinele de întreruperi înlănţuite pornesc de la adresa secundă. ***Pragma interrupt*** se utilizează obligatoriu în interiorul unei proceduri. Când compilatorul întâlneste instrucţiunea, generează automat secvenţa de salvare a regiştrilor STATUS şi FSR, plasând codul compilat din procedura respectivă la adresa vectorului de tratare a întreruperii. ***Pragma raw_interrupt*** (> jal04.55w) lasă la discreţia utilizatorului salvarea regiştrilor STATUS şi FSR.

2.8 Generarea codului

Acest paragraf dă câteva detalii despre funcționarea internă a compilatorului. Se consideră că cititorul posedă deja cunoştinţele minime despre arhitectura microcontrolerului şi că a parcurs deja documentaţia microcontrolerului cu care va lucra. Compilatorul Jal lucrează într-un număr de faze; deorece terminologia acestora provine din limba engleză, denumirile dedicate ale fazelor sunt prezentate în paranteze:

1. analiza sursei (parse)
2. prima optimizare (optimize 1)
3. conversia codului necompilabil (squash)
4. a doua optimizare (optimize 2)
5. alocarea regiştrilor (register allocation)
6. generarea codului (code generation)
7. asamblarea (assembly)
8. simulare opțională (simulate)

În faza de analiză a filei sursă (**parse**) compilatorul citeşte fila de intrare, verifică sintaxa şi semantica şi produce un arbore sintactic intern (utilizând diverşi vectori) care reprezintă sursa. Aproape toate erorile utilizator şi alte mesaje sunt generate în aceasta fază. Au loc două optimizări: expresiile constante sunt evaluate şi instrucțiunile ***if*** şi ***while*** cu o condiție constantă sunt înlocuite corespunzator. Arborele sintactic generat în faza de analiză şi tot codul corespunzator acestei faze este păstrat în memorie. Nodurile arborelui (a structurii ramificate) au o dimensiune fixă de aproximativ 40 de octeți. Pentru fiecare nod generat, locația codului sursă care a generat nodul este de asemenea memorată înafara nodului. Acest arbore intern este cauza principală pentru care compilatorul utilizează o mare cantitate de memorie (câțiva mega-octeți) pentru a compila o fila sursă de complexitate medie.

Faza de primă optimizare (**first optimize**) examinează structura ramificată şi încearcă să o transforme într-o structură ramificată echivalentă din punct de vedere semantic dar care va genera codul mai bine (mai repede şi mai compact). Variabilele, instrucțiunile şi rutinele care nu sunt utilizate, sunt şterse. Apelările înlănțuite (procedură din procedură sau procedură din funcție) sunt înlocuite cu salturi pentru a salva stiva. Unele expresii sunt înlocuite cu altele mai simple (înmulțirea este transformată în rotire, etc.) Codul şi variabilele neutilizate sunt şterse. Arborele este de asemenea simplificat pentru a-şi reduce dimensiunea şi a simplifica fazele următoare de compilare.

Faza de conversie a codului necompilabil (**squash**) înlocuieşte expresiile din arbore care nu pot fi transformate uşor în instructiuni PIC, prin construcții semantice echivalente uşor de convertit (exemplu: înmulțirile şi majoritatea rotirilor sunt înlocuite cu salturi în biblioteca de timp real. Această bibliotecă este construită în compilator şi nu are nimic de a face cu bibliotecile utilizator distribuite împreună cu compilatorul.

A doua optimizare (**second optimize**) execută acelaşi tip de optimizări ca şi prima optimizare cu excepția câtorva care ar fi încurcat execuția în faza de conversie a codului necompilabil (squash).

Faza de alocare a regiştrilor (**register allocation**) scaneaza arborele şi alocă câte o adresă fiecărei variabile care nu are încă o adresă, dar necesita una. Variabilele care nu sunt utilizate niciodată nu vor căpăta o adresă pentru că ele au fost deja şterse în fazele de optimizare.

Faza de generare a codului (**code generation**) înlocuieşte toate construcțiile în arborele sintactic cu instructiuni de tip asamblor.

Faza de asamblare (**assembly**) parcurge tot arborele sintactic şi genereaza fila asamblor şi fila hexazecimală care vor fi scrise pe disc. Dupa asamblare, se efectuează verificarea numărului de regiştrii utilizați, dimensiunea codului generat şi gradul de ocupare al stivei. Când una din aceste verificari dă o eroare, fila în limbaj de asamblare este generată în continuare utilizatorului, pentru ca acesta să o poată inspecta şi să detecteze de ce programul său utilizează atât de multe resurse; fila hexazecimală nu va fi însă generată.

Faza opțională de simulare (**simulate**), simulează codul maşină din copia internă hexazecimală şi verifică existența corectă sau eronată a egalitaților introduse în codul sursă prin **pragma test assert.**

Compilatorul utilizează o multitudine de verificari ale consistenței codului generat. Când o astfel de verificare eşuează, compilatorul va genera un mesaj de eroare şi va opri execuția. Un exemplu este faza de asamblare care verifică dacă fiecare instrucțiune în limbaj de asamblare este validă pentru microcontrolerul specificat. Opțiunea –386, face compilatorul să lucreze ceva mai repede prin renunțarea la cele mai multe verificari de acest tip.

2.8.1 Alocarea regiştrilor

In faza de alocare a regiştrilor, primul pas este de asignare a adresei biților după care sunt asignate toate adresele octeților. Regiştrii de tip bit sau octet nu sunt utilizați optim când una din ramurile arborelui foloseşte o mulțime de biți iar altă ramură nu-i foloseşte de loc. In interiorul structurii arborescente, ambele categorii de adrese sunt asignate utilizând un algoritm cu stivă fixă: compilatorul îi dă fiecarei variabile, cea mai mare adresă pe care o are disponibilă în arbore. Aceasta dă iluzia unei stive reale simple. Partea proastă este că, compilatorul poate compila doar un program complet şi nerecursiv.

2.8.2 Expresii la nivel de octet şi asignări

Expresiile de tip octet sunt evaluate în registrul w. Operatorii care nu au cod maşină corespunzători (ca de exemplu înmulțirea), sunt înlocuiți în faza de conversie (squash) cu un salt în biblioteca de timp real. Când este necesar, faza de conversie va rearanja expresiile complexe (inclusiv funcțiile) şi va însera variabile temporare. Asignarea unui octet evaluează întâi expresia din registrul w şi apoi mută valoarea în registrul corespunzător. Expresiile de tip octet şi asignările din tabelul următor sunt recunoscute ca şi cazuri speciale:

fragment de cod sursa jal	Instrucțiuni de asamblare
x = 0	clrf x
x = x + 1	incf x, f
x = x – 1	decf x, f
x = x << 1	clrc; rlf x, f
x = x >> 1	clrc; rrf x, f
x << 4	swapf y, w; andlw 0xF0
x >> 4	swapf y, w; andlw 0x0F

2.8.3 Expresii la nivel de bit şi asignări

Asignările de tip bit sunt translatate în setări de bit condiţionate şi resetări. Această structură este utilizată fie când ţinta (microcontrolerul) este volatilă (valoarea nu va fi identică pe parcursul compilării) sau expresia conţine ţinta:

```
if expression then
   target = true else target = false
end if
```

dar o structura mai compactă este de asemenea posibilă:

```
 x = false
if expression then
   target = true
end if
```

Expresiile la nivel de bit apar întotdeauna ca şi condiţii. Aceasta este o caracteristică a limbajului în cod maşina al microcontrolerului. O expresie conţinând un bit este translatată într-un set de sărituri/ignorări condiţionate, peste linii de program. Când este posibil, se preferă ignorarea următoarei linii sau o ignorare negată cu săritură. Codul generat pentru un operator va evalua operandul secund de două ori.
Următoarele asignări la nivel de bit sunt recunoscute ca speciale:

fragment de sursa jal	instrucţiuni în asamblor
b = true	bsf 31, 2
b = false	bcf 31, 2

2.8.4 Pragma jump_table

O rutină care conţine **pragma jump_table** este codată diferit în funcţie de versiunea JAL a compilatorului, în ultima pagină de memorie pentru variantele iniţiale respectiv în prima pagină pentru ultimele versiuni. Tabelul de salt pentru variabila volatilă este pus deasemenea aici. Când tabelul şi rutinele nu încap în ultima pagină de memorie, (variantele inţiale de compilator) este generat un mesaj de eroare. De asemenea când un tabel de salt sau o rutină conţinând această **pragma** este prezentă, selectarea biţilor de pagină se face automat. Jal 04.5x dispune de posibilitatea implementării unor tabele de alocare multiple, de până la 250 de caractere.

2.8.5 Pragma interrupt

Generarea codului în cazul în care nu se utilizează întreruperi, începe de la adresa 0. Când una sau mai multe întreruperi sunt prezente, codul începe cu un salt la rutina de întreruperi, iar rutinele de întreruperi înlănţuite pornesc de la adresa secundă.

Pseudo variabile şi parametri volatili. O pseudo variabilă conţine o funcţie **get** sau o procedură **put** sau pe amândouă. Utilizarea pseudo variabilelor duce la apelarea unei funcţii corespunzătoare sau a unei proceduri. Pentru fiecare variabilă care este interpretată ca şi un parametru volatil, sunt generate două salturi, una pentru **get** şi alta pentru **put.**

Indexul tabelei de salt este interpretat ca şi parametru curent. Utilizarea unui parametru volatil generează un salt la tabel, având indexul corespunzător. Valoarea din tabel este transferată spre sau dinspre procedura **get** sau **put,** utilizând o variabilă globală (valabilă peste tot în program). Compilatorul nu analizează modul de interpretare şi utilizare a parametrilor volatili. Din aceasta cauză se poate aproxima că fiecare utilizare a unui parametru volatil necesită umplerea stivei folosite de către orice procedură put sau get. De aceea utilizarea unui parametru volatil înaintea unei rutine put sau get volatile nu este permisă, deoarece poate necesita un număr infinit de intrări în stivă (după modul în care compilatorul analizează).

2.9 Biblioteci

Bibliotecile Jal se găsesc sub protecția **GNU Library General Public License**, acest lucru înseamnă că utilizatorul este liber să distribuie fie aceste biblioteci, fie biblioteci derivate din originale, dar nu poate modifica nota GPL. Această obligație nu este necesară pentru fila *.hex generată, utilizatorul având dreptul să vândă un produs care conține aceste biblioteci în forma compilată.

2.9.1 File de specificare a microcontrolerului utilizat

Aceste file conțin pragma-uri pentru cele mai comune microcontrolere PIC: 16x84, 16F87x, 16F62x, 12Fxxx, cu oscilatoare externe de 4, 10, 20 MHz respectiv 16F62x şi 12C50x, funcționând cu oscilator intern de 4MHz sau extern (conform specificației tehnice a fiecăruia) şi microcontrolerele SX18 şi SX28 cu oscilatoare externe la 50MHz sau interne de 4MHz. Deoarece toate bibliotecile conțin setările pentru câine de pază (***watchdog*)**, protecție de întârziere la alimentare (***powerup***) şi protecție la citirea memoriei (***protection***) identice cu cele de mai jos, nu sunt prezentate decât două exemple posibile:

```
pragma name 16f877_20
pragma target chip        16f877
pragma target clock       20_000_000
pragma target osc         hs
pragma target watchdog    off
pragma target powerup     on
pragma target protection  off
include jpic

pragma name 16f628_4
pragma target chip        16f628
pragma target clock       4_000_000
pragma target osc         xt
pragma target watchdog    off
pragma target powerup     on
pragma target protection  off
include jpic628
```

2.9.2 Jlib

jlib este o bibliotecă definită de utilizator. Poate să conțină următoarele biblioteci: jpic, jascii, jdelay, jseven, jstepper, jprint

Se poate observa că anumite biblioteci ca jpic sau jpic628 sunt deja incluse în filele ce definesc microcontrolerul. De asemenea dacă utilizatorul nu foloseşte motoare pas cu pas sau afişaje cu şapte segmente, bibliotecile jseven sau jstepper pot fi excluse. Este la discreția utilizatorului folosirea acestei biblioteci ca o filă de incluziune globală sau renunțarea la utilizarea ei şi definirea filelor incluse în fila sursă a proiectului. Această ultimă metodă care pare a fi cea mai bună, este utilizată în prezentarea diverselor proiecte pe care autorul cărții le-a proiectat, realizat şi testat.

2.9.3 Jpic, jpic628, jpic675

Biblioteca Jpic este interfața de bază spre resursele microcontrolelor PIC16X84, PIC16F87x şi SX. Biblioteca conține copii ale regiştrilor TRISXx si PORTx, regiştrii ce definesc direcția de comunicație a porturilor, respectiv porturile în sine. Aceste copii ajută la evitarea problemelor de comutare a paginilor de memorie (regiştrii TRISx se găsesc în bancul 1 de memorie) şi a celor de citire-modificare-scriere a pinilor individuali. Aceasta generează însă un mic surplus de cod maşină şi creşterea numărului de regiştrii utilizați. Porturile C, D şi E şi declarațiile asociate sunt implementate doar pentru PIC16F87x şi respectiv SX28 (numai portul C). Scrierea şi citirea în eeprom este implementată doar pentru microcontrolerele ce dețin memorie eeprom internă .

Biblioteca Jpic62x este specifică doar microcontrolerului PIC16F62x. Pe lângă resursele de bază ale microcontrolerului sunt implementate accesul la eeprom, transmisia serială asincronă utilizând modulul hardware USART şi câteva rutine ce accesează funcțiile analogice ale microcontrolerului. Biblioteca jpic675 este specifică microcontrolerului PIC12F675/629 şi conține suplimentar rutinele de calibrare a oscilatorului intern, de utilizare a convertorului AD sau a comparatorului intern. Pentru fiecare microcontroler nou recunoscut de compilator, se poate genera o bibliotecă jpic nouă pentru a simplifica formatul acesteia. Acest lucru este necesar deoarece deşi majoritatea regiştrilor în diversele serii de microcontrolere Microchip au aceleaşi adrese, funcțiile analogice diferă radical.

2.9.4 Regiştrii cu funcții speciale

Următorii regiştrii cu funcții speciale comuni întregii familii PIC sunt declarați în biblioteca jpic/jpic628/jpic675:

```
var volatile byte indf       at  0
var volatile byte tmr0       at  1
var volatile byte option_reg at  0x_81
var volatile byte pcl        at  2
var volatile byte status     at  3
var volatile byte fsr        at  4
var volatile byte port_a     at  5
var volatile byte tris_a     at  0x_85
var volatile byte port_b     at  6
var volatile byte tris_b     at  0x_86
```

```
var volatile byte port_c     at  7
var volatile byte tris_c     at  0x_87
var volatile byte port_d     at  8
var volatile byte tris_d     at  0x_88
var volatile byte port_e     at  9
var volatile byte tris_e     at  0x_89
var volatile byte x84_eedata at  8
var volatile byte x84_eeadr  at  9
var volatile byte pclath     at 10
var volatile byte intcon     at 11
var volatile byte option
var volatile byte trisa_s
var volatile byte trisb_s
var volatile byte trisc_s
var volatile byte trisd_s
var volatile byte trise_s
```

Variabila ***option*** nu este identică cu ***option_reg***; ea este o pseudovariabilă conținută într-o mică rutină. Variabilele ***porta…porte*** şi ***trisa…trise*** sunt copii ale regiştrilor fizici cu acelaşi nume. Următorii biți sunt conținuți în regiştrii cu funcții speciale:

```
var volatile bit  status_c    at status : 0
var volatile bit  status_dc   at status : 1
var volatile bit  status_z    at status : 2
var volatile bit  status_pd   at status : 3
var volatile bit  status_to   at status : 4
var volatile bit  status_rp0  at status : 5
var volatile bit  status_rp1  at status : 6
var volatile bit  status_irp  at status : 7
var volatile bit  intcon_rbif at intcon : 0
var volatile bit  intcon_intf at intcon : 1
var volatile bit  intcon_t0if at intcon : 2
var volatile bit  intcon_rbie at intcon : 3
var volatile bit  intcon_inte at intcon : 4
var volatile bit  intcon_t0ie at intcon : 5
var volatile bit  intcon_eeie at intcon : 6
var volatile bit  intcon_gie  at intcon : 7
```

De asemenea sunt definiți o serie de biți aparținând regiştrilor cu funcții speciale ai PIC16F87x respectiv PIC16F7x.

2.9.5 Regiştrii de direcție ai porturilor IO

Următoarele pseudo-variabile pot fi utilizate atât în stânga cât şi în dreapta instrucțiunii de alocare:

- port_a_direction, port_b_direction, port_c_direction, (octeți)
- port_a_low_direction, port_a_high_direction, port_b_low_direction, port_b_high_direction, port_c_low_direction, port_c_high_direction, (nibble= jumătate de octet)

- pin_a0_direction…pin_a4_direction, pin_b0_direction…pin_b7_direction, pin_c0_direction...pin_c7_direction, pin_d0_direction…pin_d7_direction, pin_e0_direction…pin_e2_direction, (biți)

La pornire toți pinii sunt intrări. Pentru PIC16F62x şi PIC16F87x, funcțiile analogice trebuie dezactivate (vezi biblioteca analogică janalog.jal respectiv rutinele analogice din jpic628.jal sau jpic675.jal). Următoarele constante se vor utiliza pentru a schimba direcția de comunicare a porturilor sau pinilor:

- input, output (pentru biți)
- all_input, all_output (pentru ***nibbles*** şi ***bytes***)

Pentru variabile reprezentând o jumatate de port (***nibble***) direcția este corespunzatoare cu cei mai puțini semnificativi 4 biți. Cei mai semnificativi patru biți sunt ignorați şi citiți ca 0.

2.9.6 Porturi de IO

Următoarele pseudovariabile pot fi utilizate în stânga sau în dreapta unei instrucțiuni de alocare:

- port_a, port_b, port_c, port_d, port_e, (octeți)
- port_a_low, port_a_high, port_b_low, port_b_high, port_c_low, port_c_high, port_d_low, port_d_high, port_e_low, (jumătăți de octet sau *nibble*)
- pin_a0 .. pin_a4 pin_b0 .. pin_b7, pin_c0 .. pin_c7, pin_d0…_pin_d7, pin_e0…pin_e2, (biți)

Pentru variabile de jumătate de octet valoarea este în concordanță cu cei mai puțin semnificativi 4 biți. Cei mai semnificativi 4 biți sunt ignorați şi citiți ca 0.

2.9.7 Acces indirect la regiştrii interni

Următoarele rutine sunt necesare pentru a manevra datele într-un registru specificat:

```
procedure file_get( byte in a, byte out d )
procedure file_put( byte in a, byte in d )
```

Adresele utilizate de rutinele file_get şi file_put trebuie să fie liniare, consecutive şi în spațiul de adresare al microcontrolerului. Acesta este cel mai important mod de adresare indirectă a regiştrilor din bancurile superioare de memorie unde compilatorul nu are acces direct.

2.9.8 Accesul la memoria eeprom

Următoarele rutine sunt utilizate pentru accesul datelor la o adresă specifică a memoriei eeprom:

```
procedure eeprom_get( byte in a, byte out d )
procedure eeprom_put( byte in a, byte in d )
```

Rutina eeprom_put aşteaptă ca scrierea sa fie terminată într-o buclă de aşteptare (***busy looping***).

2.9.9 Instrucțiuni speciale

```
procedure sleep
```
Procedura sleep este echivalentă cu instrucțiunea cod maşină **asm sleep** .

```
procedure clear_watchdog
```
Procedura de resetare a câinelui de pază este echivalentă cu instrucțiunea **asm clrwdt** .

```
procedure swap_nibbles( byte in out x )
```
Procedura de înlocuire a unei jumătăți de octet cu cealaltă jumătate este echivalentă cu: **asm swapf x,f .**

```
procedure bank_0
procedure bank_1
procedure bank_2
procedure bank_3
```

Aceste proceduri sunt instrucțiuni specifice de adresare indirectă a bancurilor de memorie.

```
procedure disable_comp
```
Procedură de dezactivare a comparatoarelor în microcontrolerul PIC16F62x

```
procedure no_ad
```
Procedură de dezactivare a convertoarelor AD în microcontrolerele PIC16F87x

2.9.10 jascii

Biblioteca jascii generează constantele ascii pentru caracterele ce nu pot fi tipărite:

```
const byte ASCII_NULL = 00
const byte ASCII_SOH  = 01
const byte ASCII_STX  = 02
const byte ASCII_ETX  = 03
const byte ASCII_EOT  = 04
const byte ASCII_ENQ  = 05
const byte ASCII_ACK  = 06
const byte ASCII_BEL  = 07
const byte ASCII_BS   = 08
const byte ASCII_HT   = 09
const byte ASCII_LF   = 10
const byte ASCII_VT   = 11
const byte ASCII_FF   = 12
const byte ASCII_CR   = 13
const byte ASCII_SO   = 14
const byte ASCII_SI   = 15
```

```
const byte ASCII_DLE  = 16
const byte ASCII_DC1  = 17
const byte ASCII_DC2  = 18
const byte ASCII_DC3  = 19
const byte ASCII_DC4  = 20
const byte ASCII_NAK  = 21
const byte ASCII_SYN  = 22
const byte ASCII_ETB  = 23
const byte ASCII_CAN  = 24
const byte ASCII_EM   = 25
const byte ASCII_SUB  = 26
const byte ASCII_ESC  = 27
const byte ASCII_FS   = 28
const byte ASCII_GS   = 29
const byte ASCII_RS   = 30
const byte ASCII_US   = 31
const byte ASCII_SP   = 32
const byte ASCII_DEL  = 127
```

2.9.11 jdelay

Biblioteca jdelay conține rutine de întârziere cu aşteptare (busy delay). Fiecare rutină întârzie timpul indicat de numele său înmulțit cu argumentul din paranteză. Rutinele de întârziere necesită frecvența de tact de 20MHz, 10MHz sau 4MHz. Aceste rutine au o precizie de câteva procente. Cu cât timpul necesar este mai scurt, eroarea generată este mai mare. Pentru o precizie mai mare se poate folosi fie biblioteca interval.jal fie utilizarea independentă a timerelor şi a prescalerelor interne din microcontroler pentru obținerea intervalelor necesare.

```
procedure delay_1us  ( byte in x = 1 )
procedure delay_2us  ( byte in x = 1 )
procedure delay_5us  ( byte in x = 1 )
procedure delay_10us ( byte in x = 1 )
procedure delay_20us ( byte in x = 1 )
procedure delay_50us ( byte in x = 1 )
procedure delay_100us( byte in x = 1 )
procedure delay_200us( byte in x = 1 )
procedure delay_500us( byte in x = 1 )
procedure delay_1ms  ( byte in x = 1 )
procedure delay_2ms  ( byte in x = 1 )
procedure delay_5ms  ( byte in x = 1 )
procedure delay_10ms ( byte in x = 1 )
procedure delay_20ms ( byte in x = 1 )
procedure delay_50ms ( byte in x = 1 )
procedure delay_100ms( byte in x = 1 )
procedure delay_200ms( byte in x = 1 )
procedure delay_500ms( byte in x = 1 )
procedure delay_1s   ( byte in x = 1 )
procedure delay_2s   ( byte in x = 1 )
procedure delay_5s   ( byte in x = 1 )
```

2.9.12 Jseven

Biblioteca jseven conține declarațiile pentru interfațarea cu afişaje cu 7 segmente cu LED-uri. Ambele versiuni (anod sau catod comun) sunt permise. Biblioteca include jsevenp şi asignările pinilor IO utilizați de microcontroler. Aceştia trebuie schimbați de utilizator în concordanță cu schema hardware pe care se lucrează. În biblioteca inclusă jsevenp se consideră că segmentele a…g vor fi conectate cu biții 0…6, punctul zecimal aparține bitului 7 şi fiecare segment este luminat de un nivel logic 1 (true, on). Pentru un afişaj cu anod comun se va utiliza funcția jseven negată. Următoarele constante (din jsevenp) definesc segmentele individual:

```
const byte seven_segment_a
const byte seven_segment_b
const byte seven_segment_c
const byte seven_segment_d
const byte seven_segment_e
const byte seven_segment_f
const byte seven_segment_g
const byte seven_segment_dp
```

Următoarele constante definesc imaginea obținută pentru valorile 0…15 şi spațiu:

```
const byte seven_space
const byte seven_value_0
const byte seven_value_1
const byte seven_value_2
const byte seven_value_3
const byte seven_value_4
const byte seven_value_5
const byte seven_value_6
const byte seven_value_7
const byte seven_value_8
const byte seven_value_9
const byte seven_value_a
const byte seven_value_b
const byte seven_value_c
const byte seven_value_d
const byte seven_value_e
const byte seven_value_f
```

Următoarea rutină întoarce valoarea afişajului cu şapte segmente pentru valoarea corespunzătoare argumentului x:

```
function seven_from_digit( byte in x )  return x
```

2.9.13 Jstepper

Biblioteca jstepper conține rutinele pentru motoare unipolare cu patru faze. Rutinele sunt:

```
procedure stepper_motor_full_forward( byte in out x )
procedure stepper_motor_half_forward( byte in out x )
```

```
procedure stepper_motor_full_backward( byte in out x )
procedure stepper_motor_half_backward( byte in out x )
```

Biblioteca jstepern (CD:/tools/jal_compiler/extra_libraries) conține și rutinele full power:

```
procedure stepper_motor_power_forward ( byte in out x )
procedure stepper_motor_power_backward( byte in out x )
```

Bobinele motoarelor sunt activate de un nivel logic high. Rutina cu pași întregi conține mai puțin cod decât rutina cu pas pe jumătate. De reținut că puterea la axul motorului în modul jumătate de pas, `half_backward/forward` respectiv în modul `power`, este mai mare decât în modul pas întreg. Numai cei mai puțin semnificativi patru biți ai octetului x trebuie utilizați, biții semnificativi sunt ignorați și vor conține 0 la ieșirea din rutină.

2.9.14 Jprint

Biblioteca jprint conține rutine care printează o valoare în diverse baze de numerație. Fiecare rutină are același argument. Rutinele sunt:

```
 procedure print_binary_8(
   byte volatile out target,
   byte in x,
   byte in leader = "0" )

procedure print_binary_4( ... )
print_decimal_3( ... )
print_decimal_2( ... )
print_decimal_1( ... )
print_hexadecimal_2( ... )
print_hexadecimal_1( ... )
```

Argumentul numit **target** este destinația de ieșire . Aceasta trebuie să fie o pseudo-variabilă (procedură put) care poate manipula scrieri succesive. Argumentul **x** este valoarea care va fi printată. Procedura care nu va printa întregul argument, va printa numai numărul de digiți mai puțin semnificativi indicat de numele procedurii. **Leader**-ul este valoarea ASCII care este printată în locul cifrei 0. Implicit este "0" (0 ca simbol ASCII). Aceasta cauzează printarea simbolului ASCII 0 în câmpurile care nu sunt ocupate de rezultat. Un 0 binar va suprima printarea zerourilor ASCII. Introducerea spațiului (blank) ca leader este eficientă când este necesară suprimarea zerourilor nesemnificative (de exemplu afișarea numărului 0196 se transformă în _196, unde _ reprezintă afișaj stins.

Procedura **print_decimal_3** are uneori un comportament anormal dacă se încearcă printarea unor valori mai mari de 255, ca rezultat al unei operații matematice anterioare. Utilizatorul va folosi aceasta procedură cu precauție, în situația în care nu functionează corect, ea poate fi înlocuită cu **print_hexadecimal_2** urmată de o conversie **bin_bcd** (vezi biblioteca matematică) sau de **send_lcd_3** (CD:/tools/jal_compiler/extra_libraries/print).

2.9.15 Interval

Aceasta bibliotecă este suportată numai de microcontrolere PIC16F62x, PIC16X84, PIC16F87x

Biblioteca interval conține rutine de întârziere bazate pe întreruperi generate de timerul tmr0. Primul interval necesar T, trebuie pregătit prin apelarea procedurii **init_interval**. Intervalul de timp T care este inițializat, este egal cu timpul indicat de numele procedurii, înmulțit cu argumentul. Din acest moment, un interval de timp expiră la fiecare multiplu de timp T, după ce procedura **init_interval** a fost apelată. O apelare a procedurii **next_interval** se va termina la următoarea expirare a intervalului inițiat. Primul interval de după apelarea procedurii **init_interval** poate dura ceva mai mult decât T. Când o apelare a rutinei **next_interval** are loc după ce durata intervalului a trecut deja, apelarea poate dura echivalentul unui argument egal cu 255, corespunzator procedurii **init_interval.** Biblioteca de întârziere delay, utilizează o rutină de întreruperi, tmr0, prescalerul şi utilizează din timpul microcontrolerului execuția a 17 instrucțiuni cod maşina, 5 regiştrii şi un acces la stivă. Frecvența de tact a procesorului trebuie să fie 10MHz sau 4MHz. Următoarele rutine aparțin acestei biblioteci:

```
procedure init_interval_1uS  ( byte in n = 1 )
procedure init_interval_2uS  ( byte in n = 1 )
procedure init_interval_5uS  ( byte in n = 1 )
procedure init_interval_10uS ( byte in n = 1 )
procedure init_interval_20uS ( byte in n = 1 )
procedure init_interval_50uS ( byte in n = 1 )
procedure init_interval_100uS( byte in n = 1 )
procedure init_interval_200uS( byte in n = 1 )
procedure init_interval_500uS( byte in n = 1 )
procedure init_interval_1mS  ( byte in n = 1 )
procedure init_interval_2mS  ( byte in n = 1 )
procedure init_interval_5mS  ( byte in n = 1 )
procedure init_interval_10mS ( byte in n = 1 )
procedure init_interval_20mS ( byte in n = 1 )
procedure init_interval_50mS ( byte in n = 1 )
procedure init_interval_100mS( byte in n = 1 )
procedure init_interval_200mS( byte in n = 1 )
procedure init_interval_500mS( byte in n = 1 )
procedure init_interval_1S   ( byte in n = 1 )
procedure next_interval
```

Utilizarea acestei biblioteci nu permite crearea unei rutine de intreruperi la discreția utilizatorului, decât prin modificarea corespunzătoare a bibliotecii interval.jal.

2.9.16 Hd447804, Hd447808

Aceste biblioteci asigură interfațarea pe 4 biți (6 pini) şi 8 biți (10 pini) la controlerul LCD Hitachi HD44780. Amândouă bibliotecile includ hd44780p, bibliotecă care conține asignarea pinilor microcontrolerului. Această bibliotecă poate fi adaptată de utilizator (cu nume schimbat, pentru a avea referința de model) în funcție de aplicația hardware. Rutinele incluse sunt:

```
procedure hd44780_clear
procedure hd44780_position1( byte in x )
procedure hd44780_position2( byte in x )
procedure hd44780_line1
procedure hd44780_line2
procedure cursor_blink ( byte in x )
procedure cursor_off
procedure cursor_left
procedure cursor_right
procedure shift_left
procedure shift_right
procedure hd44780_write( byte in x )
var byte volatile hd44780
procedure hd44780_define(
   byte in x,
   byte in d0,
   byte in d1,
   byte in d2,
   byte in d3,
   byte in d4,
   byte in d5,
   byte in d6,
   byte in d7
)
```

hd44780_clear şterge afişajul şi pune cursorul în linia întâia poziţia 0
hd44780_position1 pune cursorul la poziţia indicată în linia 1 fără să şteargă afişajul
hd44780_position2 pune cursorul la poziţia indicată în linia 2 fără să şteargă afişajul
hd44780_line1 şi **hd44780_line2** pun cursorul la începutul linei întâi sau doi fără să şteargă afişajul
procedure cursor_blink pentru x=1 caracterul şi cursorul pâlpâie, pentru x=2 cursorul este afişat, pentru x=3 cursorul este afişat şi caracterul corespunzator pâlpâie
procedure cursor_off stinge cursorul curent
procedure cursor_left muta cursorul curent cu o poziţie la stânga
procedure cursor_right muta cursorul cu o poziţie la dreapta
procedure shift_left curge întregul text la stânga
procedure shift_right curge întregul text la dreapta
hd44780_write scrie caracterul indicat la poziţia curentă şi avansează cursorul
Asignarea unui caracter la instrucţiunea **hd44780** are acelaşi efect ca şi o apelare a rutinei **hd44780_put** .

O apelare a rutinei **hd44780_define,** defineşte imaginea caracterului cu adresa x. X trebuie să fie într-un domeniu cuprins intre 0 .. 7. Octeţii b0…b7 definesc fiecare un rând al imaginii. B0 defineşte rândul de sus, b7 defineşte rândul de jos. Bitul cel mai puţin semnificativ (0), defineşte pixelul din dreapta, un nivel logic 1 (on, high) marchează pixelul ca întunecat în cazul unui afişaj cu reflexie. Caracterul este definit de o matrice de 5x8 pixeli, bitul 4 defineşte pixelul aflat cel mai în stânga caracterului.
exemplu:

```
include 16f84_10
include jlib
include hd447804
```

```
 hd44780_define ( 2,
-- adresa caracterului cuprinsă între 0 şi 7
                0b_0000_0110,
                0b_0000_1001,
                0b_0000_1001,
                0b_0000_0110,
                0b_0000_0000,
                0b_0000_0000,
                0b_0000_0000,
                0b_0000_0000 )
   hd44780_clear
   hd44780_line1 -- rândul 1
   hd44780 = "G" hd44780 = "r" hd44780 = "a" hd44780 = "d"
   hd44780_position2 ( 0 )  -- rândul 2
   hd44780 = "C" hd44780 = "e" hd44780 = "l" hd44780 = "s"
   hd44780 = "i" hd44780 = "u" hd44780 = "s" hd44780 = "="
   hd44780_position2 ( 8 )
   hd44780 = 2
-- scrie semnul corespunzator gradului Celsius
   hd44780 = "C"
```

2.9.17 i2c

Această bibliotecă asigură procedurile pentru operarea i2c prin software. Biblioteca include i2cp, o bibliotecă inclusă ce conţine asignările pinilor IO. O copie a acestei biblioteci poate fi adaptată de utilizator conform cerinţelor sale. Următoarele rutine de bază i2c sunt necesare pentru a construi protocolul i2c:

```
   procedure i2c_put_start
   procedure i2c_put_put_read_address( byte in a )
   procedure i2c_put_write_address( byte in a )
   procedure i2c_put_ack
   procedure i2c_put_nack
   procedure i2c_wait_ack
   procedure i2c_put_byte( byte in d )
   procedure i2c_get_byte( byte out d )
   procedure i2c_put_stop
   procedure i2c_put_nack_stop
```

Aceste rutine nu sunt necesare când se utilizează circuite integrate care comunică pe bus I2C hardware.

2.9.17.1 Protocolul i2c

Următoarele rutine asigură buna funcţionare a protocolului i2c prin metoda software:

```
procedure_i2c_read_1( byte in a, byte out d )
procedure i2c_write_1( byte in a, byte in d )
procedure i2c_read_2( byte in a, byte out d1, byte out d2 )
procedure i2c_write_2( byte in a, byte in d1, byte in d2 )
```

2.9.18 Lm75

Această bibliotecă conține rutinele de interfațare a senzorului de temperatură LM75 la microcontroler. Biblioteca include biblioteca i2c, care la rândul ei conține biblioteca i2cp unde sunt definiți pinii de intrare-ieşire. O copie locală a acestei biblioteci poate fi adaptată pentru a corespunde nevoilor utilizatorului. Următoarele rutine lm75 sunt conținute în bibliotecă:

```
procedure lm75_read_raw( byte  in address,
                         byte out d1,  byte out d2 )

 procedure lm75_read_fdt(
   byte   in address,
   bit   out freezing,
   byte  out degrees,
   byte  out tenth )
```

Procedura **lm75_read_raw** returnează cei doi octeți de date citiți din registru de temperatură a lui LM75. Procedura **lm75_read_fdt** returnează informația de temperatură conținută în trei variabile:

- **freezing** indică dacă temperatura este negativă
- **degrees** este temperatura absolută în grade Celsius
- **tenth** este zecimala de grad Celsius

Nota: comunicația i2c software (clock unidirecțional pentru master-slave şi data bidirecțională) implică utilizarea a două rezistențe de pull-up pe pinul de date şi clock.

2.9.19 serial

Această bibliotecă nu suportă microcontrolere Scenix SX18 si SX28.

Biblioteca conține rutine de transmisie şi recepție serială cu aşteptare (busy-waiting). Are inclusă **serialp**, o bibliotecă ce conține definirea pinilor de comunicație, rata de transfer a datelor şi polaritatea acestora. O copie a acestei biblioteci poate fi utilizată pentru a adapta comunicația cu nevoile utilizatorului. Rutinele conținute în bibliotecă sunt:

```
asynch_send( byte in x )
var byte volatile asynch
asynch_receive( byte out x )
asynch_poll( byte out x )  return bit
```

O apelare a procedurii **asynch_send** trimite octetul x pe linia serială.
Asignarea prescurtată **asynch** are acelaşi efect ca şi apelarea procedurii **asynch_send**.
Apelarea procedurii **asynch_receive** are ca rezultat întoarcerea octetului receptionat în x. Apelarea aşteaptă până când un octet este recepționat. Daca recepția nu este fluentă, se pierde timp valoros în aşteptare. O apelare a procedurii **asynch_poll** returnează octetul receptionat în x. Din procedură se sare rapid în programul principal când nu se poate recepționa nici un octet. Rezultatul funcției returnează un bit care indică dacă s-a recepționat un octet sau nu.

```
include 16f84_10
```

```
   include jlib
   include serial
asynch = "H" asynch = "e" asynch = "l" asynch = "l" asynch = "o"
asynch = " " asynch = "W" asynch = "o" asynch = "r" asynch = "l"
asynch = "d"
asynch = ASCII_CR asynch = ASCII_LF
```

Notă: utilizarea acestei biblioteci poate fi problematică la viteze mai mari de 9600bps existând situaţii în care comunicaţia între două microcontrolere sau între un microcontroler şi un PC nu funcţionează corespunzător .

2.9.20 Random3

Biblioteca random3 generează biţi şi octeţi în mod pseudo-aleator utilizând un registru liniar de deplasare cu reacţie de 24 de biţi. Registrul de deplasare care generează date pseudoaleatoare nu este iniţializat automat. Acest lucru poate fi benefic sau nu în funcţie de aplicaţia utilizatorului. Valoarea iniţială a registrului nu poate fi foarte aleatoare.

```
procedure randomize( byte in n )
```

O apelare a procedurii iniţializeaza registrul FSR cu valoarea n şi realizează o deplasare de 24 de ori. Acest lucru dă un punct de start aleator care poate genera un octet pseudo-aleator. Când procedura startează cu aceeaşi valoare constantă ca parametru, se va obţine aceeaşi secvenţă pseudo-aleatoare.

```
function random_bit return bit
```

Aceasta funcţie returnează următorul bit pseudo-aleator.

```
function random_byte return byte
```

Aceasta funcţie returnează următorul octet pseudo-aleator.

2.9.21 Cio

Biblioteca cio (Chained IO = înlănţuire) creează o posibilitate de a extinde aproape la infinit numărul de intrări şi ieşiri ai microcontrolerului utilizând regiştrii de ieşire in serie. Sunt utilizaţi în bibliotecă patru regiştrii de ieşire cu intrare serială şi ieşire paralelă şi patru regiştrii cu intrare paralelă şi ieşire serială. Pentru conectarea acestor regiştrii sunt necesari 6 pini ai microcontrolerului: clock, data, şi load pentru ambele înlănţuiri de regiştrii de intrare-ieşire. Cu o utilizare multiplexată aceşti 6 pini pot fi reduşi la doar 3. Biblioteca include ciop, o bibliotecă ce conţine asignarea pinilor şi alte câteva opţiuni. O copie a acestei biblioteci poate fi adaptată de utilizator după dorinţă.

2.9.21.1 Teoria transferului

Datele de ieşire sunt rotite serial cu umplerea regiştrilor de deplasare în sens invers: data pentru cel mai semnificativ registru comandat iese prima, urmată de data pentru următorul registru de deplasare, şamd. Când toţi regiştrii sunt încărcaţi, o comandă de

încarcare paralelă este generată pentru a transfera datele din regiştrii la pinii de ieşire. Datele de intrare sunt mai întâi încărcate şi apoi rotite serial spre intrarea PIC-ului. Datele aparţinând registrului din imediata vecinatate a microcontrolerului sunt transferate primele. Numarul maxim de regiştrii de deplasare ce pot fi înlănţuiţi este dependentă de puterea de calcul a microcontrolerului, problemele de alunecare a tactului generat şi de cea mai importantă problemă: timpul necesar pentru încărcarea tuturor regiştrilor din lanţ.
In funcţie de sarcina conectată pe pinii PIC-ului (fan-out) şi de capacitaţile parazite ale conexiunilor, poate fi necesară o întârziere suplimentară în durata tactului, aceasta creşte timpul necesar pentru a încărca datele în regiştrii. Această întârziere poate fi specificată în fila de configurare a comunicaţiei.

Schema electronică şi cablajul trebuie proiectate pentru a preveni alunecarea sau oscilaţia tactului: întârzierile mari şi tranziţiile asincrone ale intrărilor regiştrilor pot produce încărcări false ale regiştrilor pe fronturi parazite, astfel se pot pierde biţi. Pentru a evita astfel de situaţii, se recomandă utilizarea aceluiaşi tip de registru de deplasare pentru realizarea întregului lanţ, utilizarea unui buffer pentru obţinerea unui tact curat cu tranziţii rapide sau utilizarea unor regiştrii de deplasare (ca 4094) cu ieşire întârziată. Când sunt permise impulsuri parazite scurte (glitch-uri) la ieşirea regiştrilor (un exemplu este utilizarea LED-urilor la ieşire) sau încărcarea paralelă în regiştrii poate fi activă permanent, poate fi utilizat un registru de deplasare mai ieftin (74164). In ambele cazuri, sarcina care încarcă PIC-ul fiind transferată pe aceşti regiştrii, un număr mai mare de pini ai PIC-ului pot fi utilizaţi pentru alte scopuri. Este posibilă multiplexarea pinilor de comandă ai regiştrilor de intrare ieşire (load, clock, data) cu alţi pini utilizaţi pentru alte dispozitive periferice (ca de exemplu liniile de date a afişajului LCD, cu dezactivarea acestuia pe parcursul comunicaţiei). Proiectantul trebuie să ia în considerare că starea iniţială a regiştrilor este necunoscută. Pentru regiştrii de deplasare ce au reset, acesta poate fi activat în faza de alimentare sau ori de câte ori aplicaţia o cere. Cu configuraţia de bază (fără întârzieri suplimentare) pentru un PIC16F84 funcţionând la 10MHz, apelarea rutinei cio_out_8_load sau a rutinei cio_load_8_in, durează aproximativ 100 uS .

2.9.21.2 Configuraţia

Fila de configurare defineşte tipul de lanţ de regiştrii ce este utilizat (de intrare sau/şi de ieşire), denumirea şi polaritatea pinilor microcontrolerului, opţional întârzierea dintre tranziţii şi modul de utilizare al încărcării paralele, sau multiplexarea funcţiilor pinilor de tact şi/sau de încărcare a regiştrilor. Unii dintre cei mai utilizaţi regiştrii de deplasare pentru ieşiri sunt:

- 74LS595, 74164, registru de deplasare serial de 8 biţi cu acţionare pe front (TTL, HCT, HS, LS) şi CD4094 (CMOS),

respectiv pentru intrări:

- 74165, registru asincron de 8 biţi cu încărcare paralelă şi deplasare serială,
- 74166 registru sincron de 8 biţi cu încărcare paralelă şi deplasare serială (TTL, LS).

Configuraţiile implicite sunt setate pentru regiştrii 74HCT595, HEF4094 (ieşiri) respectiv 74LS166 (intrări). Fila de configurare trebuie adaptată pentru circuite care necesită o polaritate diferită, respectiv încărcarea sau tactul registrului diferite.

2.9.21.3 Interfața de nivel scăzut

```
procedure cio_out_load
procedure cio_out_byte( byte in data )
procedure cio_in_load
procedure cio_in_byte( byte out data )
```

Procedurile **cio_out_byte** şi **cio_out_load** pot fi utilizate pentru a încărca lanțul de regiştrii (registrul cel mai semnificativ primul) şi pentru a trimite datele pe pinii de ieşire ai regiştrilor. Procedurile **cio_in_load** şi **cio_in_byte** pot fi utilizate pentru a încărca şi a citi lanțul de regiştrii cu datele de intrare (registrul cel mai apropiat este citit primul).

2.9.21.4 Interfața de nivel ridicat

```
procedure cio_out_1_load( byte in d1 ) …
procedure cio_out_8_load( byte in d1, … byte in d8 )
procedure cio_load_1_in( byte out d1 ) …
procedure cio_load_8_in(byte out d1, … byte out d8 )
```

Una din procedurile **cio_out_N_load** (N = 1 ... 8) poate fi utilizată pentru transferul a N octeți de date şi ieşire a datelor spre sarcină. Primul parametru al procedurii **cio_out_N_load** este octetul destinat registrului de deplasare cel mai apropiat de PIC. Una din procedurile **cio_load_N_in** poate fi utilizată pentru a încărca şi a transfera N date de intrare. Primul parametru a unei proceduri **cio_load_N_in** este octetul provenit din registrul de deplasare cel mai apropiat de PIC .

2.10 Jal în doar câteva cuvinte

Jal este un limbaj de nivel înalt orientat la nivel de blocuri. Seamană cu limbajul Pascal dar poate fi denumit “ADA pentru microcontrolere” sau “BASIC structurat”. Tipurile recunoscute de Jal sunt de tip bit şi de tip octet pentru faza de rulare şi 32 de biţi de tip întregi-universali pentru faza de compilare. Instrucţiunile sunt asemănătoare cu cele scrise în C dar o asignare (atribuire de valoare) poate fi considerată o instrucţiune. Operatorii existenţi sunt: + - / * % ! & | ^ < < == != < = sau =. Ordinea prioritaţilor este asemănătoare limbajului C şi parantezele pot fi utilizate pentru grupări. Nu există restricţii privitor la complexitatea expresiilor. Utilizatorul poate defini operatorii proprii, dar priorităţile sunt fixe şi operatorii existenţi nu pot fi redeclaraţi. Variabilele trebuie declarate înainte de utilizare şi pot fi legate de o adresă particulară pe care o doreşte utilizatorul sau alocate automat de compilator utilizând algoritmul cu stivă fixă. O declarare de variabilă poate apare oriunde în program, la fel ca şi o instrucţiune. Variabila este activă din momentul declarării şi până la sfârşitul blocului ce o include. Procedurile şi funcţiile pot avea parametrii. Fiecare parametru are un nume, un tip, un mod (intrare, ieşire sau combinat) şi opţional o valoare implicită. Interpretarea parametrilor se face fie după valoarea de referinţă fie după rezultatul obţinut la terminarea procedurii în funcţie de modul care se potriveşte compilatorului cel mai bine (uzual este după valoarea rezultatului, excepţie făcând parametrii volatili care sunt interpretaţi ca pointeri de acces în rutinele corespunzătoare). Un parametru poate avea o valoare implicită, caz în care nu este nevoie ca să existe o valoare curentă (actuală) pentru acel parametru. Funcţiile pot fi utilizate în expresii, procedurile pot fi utilizate ca instrucţiuni de sine stătătoare. Când nici un parametru nu apare în interiorul parantezelor procedurii, aceastea pot fi omise. Instrucţiunile sunt: asignarea, *if-then-elsif-else*, *for*, *loop* şi procedurile apelate. Partea *else* a unei proceduri *if-then-else* este optională. O bucla de tip *loop* poate avea un număr (for 10 loop ...), o condiţie (while ... loop ...) sau poate fi necondiţionată (forever loop ...).
Rutinele *put* şi *get* pot fi utilizate pentru a construi interfeţe care se utilizează ca şi variabilele. Această metodă este utilizată în biblioteca jpic pentru a ascunde valoarea bufferului portului respectiv, ascundere necesară pentru evitarea problemelor de citire-modificare-scriere existente în arhitectura PIC. Limbajul de asamblare în linie este suportat utilizând sintaxa Microchip. Un simulator integrat este conţinut în microcontroler, pentru a testa compilatorul şi codul generat. Compilatorul genereaza atât fila *.hex care poate fi transferată direct în memoria microcontrolerului utilizând un programator standard, cât şi fila *.asm care poate fi utilizată de către sculele de dezvoltare oferite de Microchip.

2.11 Exemple

2.11.1 e0001 : LED care pulsează

Următorul program pulsează un LED conectat pe pinul A0 printr-o rezistenţă corespunzătoare conectată către VCC sau către GND. Rezistenţa se dimensionează utilizând legea lui Ohm, cunoscând căderea de tensiune medie pe LED de cca 1.2V…1.5V (dependentă de culoarea LED-ului). LED-ul se va monta cu anodul la VCC şi catodul la rezistenţa conectată pe pinul A0 sau cu anodul la rezistenţa conectata la pinul A0 şi catodul la GND. Pentru un LED standard de 5mm diametru, catodul este marcat de o dungă (obţinută la turnarea materialului plastic) pe corpul LEDului.

```
 [1]  -- pulsează un LED pe pinul A0
 [2]  include 16f84_10
 [3]  include jlib
 [4]  pin_a0_direction = output
 [5]  forever loop
 [6]     pin_a0 = on
 [7]     delay_1s
 [8]     pin_a0 = off
 [9]     delay_1s
[10]  end loop
```

[1] Jal este un limbaj cu un format de scriere liber. Sfârşitul liniei nu are nici un scop precis, excepţie făcând comentariile. Un comentariu începe cu două minusuri (--) sau cu punct şi virgulă (;). **Numerele incluse în paranteze drepte [1] sunt utilizate doar pentru explicaţii, ele nu fac parte din program !**
[2] Microcontrolerul este PIC16F84 cu un cuarţ de 10MHz (poate fi utilizat şi PIC16F84A ce acceptă tact de 20MHz dar cu un cuarţ de 10 MHz, sau orice alt microcontroler, cu condiţia ca fila de definire să fie modificată corespunzător)
[3] Biblioteca standard jlib este inclusă aici
[4] La pornire toţi pinii sunt intrări, această instrucţiune face pinul A0 sa fie ieşire.
[5] Partea principală a programului este o buclă fără de sfârşit
[6] Pinul A0 este setat in 1 logic. **High** şi **on** sunt sinonime pentru **true** adică 1 logic, pe când **low** şi **off** sunt sinonime pentru **false**, adica 0 logic. Ieşirile (**output)** şi intrările (**input)** sunt declarate în biblioteca jpic, inclusă în jlib. Rutinele de ieşire din biblioteca jpic utilizează un buffer al portului de ieşire pentru a evita problemele de citire-modificare-scriere existente in arhitectura PIC.
[7] Această procedură apelează o rutină de intârziere de 1 secundă. Există şi alte proceduri înrudite care asigură întârzieri de 100mS, 10mS, 1mS şi 100μS sau multiplii ai acestora. Argumentul unei proceduri de întârziere (delay) este un octet, deci domeniul de validitate este 0…255. Toate calculele în jal sunt făcute modulo 256. Argumentul implicit este 1, **delay_1S (1),** deci instrucţiunea va cauza o întârziere de 1 secundă. Rutinele de întârziere necesită specificarea clară a frecvenţei de tact la care PIC-ul lucrează. Dacă vom include biblioteca 16f84_4 şi vom utiliza un oscilator de 10MHz valoarea reală a întârzierii va fi de 2.5 secunde (10MHz/4 = 0.25 μS, 4MHz/4 = 1 μS, frecvenţa cuarţului este divizată intern cu 4, vezi foaia de catalog a microcontrolerului).
[8] Pinul A0 este setat în 0 logic
[9] Aceeaşi funcţie ca linia [7]: o întârziere de o secundă
[10] Aceasta linie indică sfârşitul buclei începute de linia [5].

2.11.2 e0002 : călăreţ în noapte cu LED-uri

Următorul program demonstrează funcţionarea unui călăreţ în noapte cu LED-ri pe portul B. Călăreţul în noapte se utilizează pentru semnalizarea maşinilor cu gabarit mare dar şi a autoturismelor pe timp de noapte. El se montează pe luneta/parbrizul autovehiculului avertizând prin mişcarea succesiva a luminii stânga-dreapta, pe conducătorii autovehiculelor din spatele sau din faţa vehicolului în cauză.

```
[ 1]   include 16f84_10
[ 2]   include jlib
[ 3]
[ 4]   -- călăreţ în noapte cu LED-uri pe portul B
[ 5]   const bit to_right  = high
[ 6]   const bit to_left   = low
[ 7]
[ 8]   procedure night( byte in out x,
                       bit in out direction )  is
[ 9]      if ( x & 0b_1000_0000 )  != 0 then
[10]         direction = to_right
[11]      end if
[12]      if ( x & 0b_0000_0001 )  != 0 then
[13]         direction = to_left
[14]      end if
[15]
[16]      if direction == to_right then
[17]         x = x >> 1
[18]      else
[19]         x = x << 1
[20]      end if
[21]   end procedure
[22]  -- toţi pinii portului b sunt ieşiri
[23]  port_b_direction = all_output
[24]  var byte x = 0b_0000_0001
[25]  var byte d = to_left
[26]  -- bucla principală
[27]  forever loop
[28]    -- seteaza portul b
[29]    -- pentru LED-uri active pe 0 înlocuieşte
[30]    port_b = x -- cu port_b = x ^ 0xFF
[31]    -- delay 200mS
[32]    delay_100ms( 2 )
[33] -- deplaseaza x un pas în direcţia indicată de d
[34]    night( x, d )
[35]  end loop
```

[5] jal04.xx nu suportă variabilele string (şiruri alfanumerice), de aceea două constante de tip bit sunt utilizate pentru identificare direcţiei curente în care se mişca LED-urile.
[8] Este declarată procedura *night*. Aceasta are doi parametrii de intrare-ieşire: afişarea şi direcţia curentă de mişcare. Parametrii cu funcţie dublă de intrare-ieşire sunt copiaţi în şi din parametrii curenţi. Parametrii de intrare sunt copiaţi numai înaintea execuţiei procedurii iar parametrii de ieşire sunt copiaţi numai după ce procedura a fost executată.
[9,16] După ce valoarea curentă afişată a atins marginea stângă sau dreaptă, direcţia curentă de deplasare este rememorată cu noua valoare. Valoarea afişată este prelucrată cu ŞI logic cu 0b_0000_0001 pentru a detecta o atingere a marginii drepte şi cu 0b_1000_0000 pentru a detecta o atingere a marginii stângi a direcţiei de mişcare. Prefixul 0b indică faptul că este vorba de cod binar. Celelalte prefixe utilizate: 0h indică codul hexazecimal iar 0d indică codul zecimal care este implicit. Liniile de separaţie între grupele de patru biţi şi prefix sunt ignorate de compilator.
[16] In funcţie de valoarea curentă a bitului **direction,** valoarea afişată este deplasată cu o poziţie spre stânga sau spre dreapta.

[23] Această instrucţiune trece toţi pinii portului B ca ieşiri. Funcţia **all_output** este declarată în biblioteca jpic, inclusă în jlib.
[24] Valoarea iniţială a afişajului cu LED-uri este 0b_0000_0001. O valoare iniţială 0b_0000_0101 sau 0b_0000_1111 va produce de asemenea un efect spectaculos.
[25] Direcţia iniţială este spre stânga (**to_left**). Aceasta poate fi omisă deoarece valoarea iniţială a lui **x** face ca direcţia să fie stabilită imediat.
[27] Bucla infinită setează ultima valoare pe portul b, aşteaptă 200mS şi apelează procedura **night** pentru a calcula următoarea valoare afişată şi posibila schimbare de direcţie.
[30] pentru LED-uri active pe 0 logic, această linie poate fi modificată pentru a inversa valoarea lui x înainte ca aceasta să fie alocată portului b: `port_b = x ^ 0xFF.`
Schema electronică este prezentată în figura de mai jos:

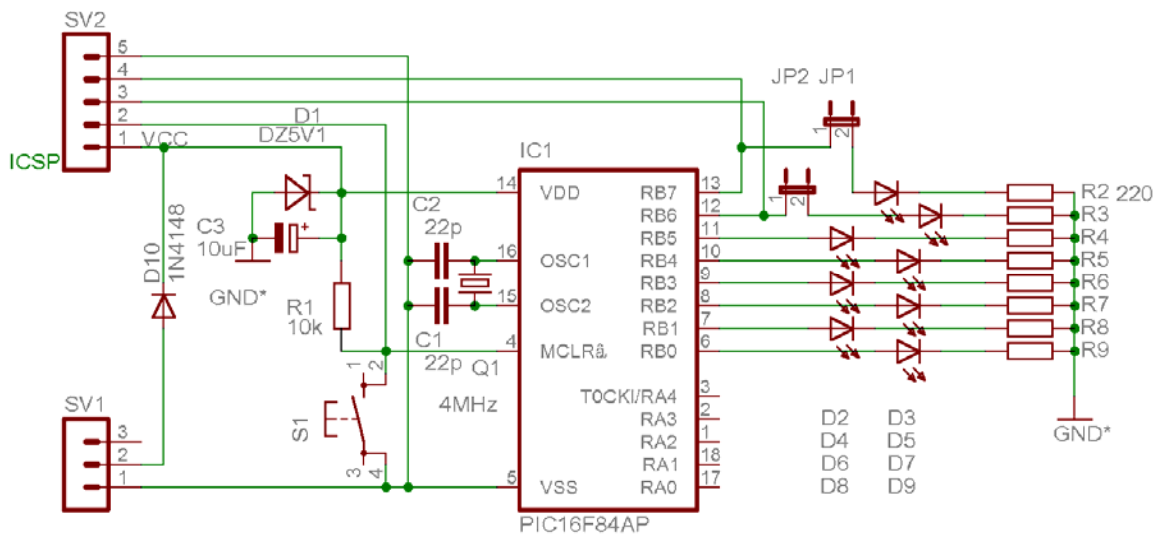


fig. 2-1 Călăreţ în noapte pe portul B

SV1 este conectorul de alimentare. O diodă "antiprost" D10 este utilizată pentru protecţia microcontrolerului la tensiune de alimentare inversă iar D1 asigură protecţia împotriva tensiunilor de alimentare aplicate accidental, mai mari decât +5V (protecţia este activă doar o perioadă scurtă de timp, până la distrugerea diodei, deoarece în schemă nu este prezentă nici o rezistenţă de limitare a curentului prin circuit, utilizatorul trebuie să asigure tensiunea de alimentare de +5V conform specificaţiei tehnice a microcontrolerului). Butonul S1 asigură resetarea manuală a cipului. Grupul R2…R9 poate fi înlocuit cu o reţea rezistivă, LED-urile pot fi de tip "*superbright*" de 5mm sau 10 mm diametru.

Doi jumperi JP1 şi JP2 (ambii în pozitia deschis) asigură programarea în circuit a microcontrolerului de către orice tip de programator. Utilizatorul poate verifica faptul că programatorul paralel descris în cap.1 nu necesită deconectarea jumperilor, deoarece poate asigura atât curentul de programare cât şi curentul de aprindere al LED-urilor D2 şi D3 care, pe perioada de programare vor semnaliza starea 1 logic a datelor şi a tactului. După ce codul hexa a fost scris, jumperii trec în poziţia închis pentru a verifica buna funcţionare a programului. Conectorul SV2 este destinat programării ICSP (In Circuit Serial Programming).

2.11.3 e0003 : robot care urmăreşte o linie

Următorul program controlează un robot simplu care se deplasează dealungul unei linii trasate pe o pardosea. Linia trebuie să aibă o lăţime suficientă şi un contrast bun raportată la pardosea. Robotul se deplasează utilizând două motoare pas cu pas recuperate din unităţi vechi de floppy-disk de 5,25 inch şi doi senzori de infraroşu reflectivi. Robotul urmareşte o linie neagră trasată pe un fundal alb alimentând fiecare motor când senzorul asociat vede alb. Cu modificari minore robotul poate fi făcut să urmărească o linie albă trasată pe un fundal negru.

```
[ 1]   ; Robot care urmăreşte o linie
[ 2]   ; portulB alimentează două motoare unipolare prin ULN2803
[ 3]   ; a0 si a1 sunt conectaţi la 2 senzori reflectivi (alb=low)
[ 4]   ; a2 si a3 alimenteaza 2 LED-uri care arată starea
       ; senzorilor
[ 5]
[ 6]   include 16f84_10
[ 7]   include jlib
[ 8]
[ 9]   port_b_direction = all_output
[10]   pin_a0_direction = input
[11]   pin_a1_direction = input
[12]   pin_a2_direction = output
[13]   pin_a3_direction = output
[14]
[15]   procedure steppers( byte in a, b )  is
[16]      port_b = a + ( b < <  4 )
[17]      delay_1mS( 10 )
[18]   end procedure
[19]
[20]   var byte left_stepper  = 0b_0001
[21]   var byte right_stepper = 0b_0001
[22]
[23]   forever loop
[24]      pin_a2 = pin_a0
[25]      pin_a3 = pin_a1
[26]
[27]  if ! pin_a0 then
[28]    stepper_motor_half_forward( right_stepper )
[29]  end if
[30]     if ! pin_a1 then
[31]      stepper_motor_half_forward( left_stepper )
[32]      end if
[33]
[34]      steppers( left_stepper, right_stepper )
[35]   end loop
```

[15] Procedura **steppers** are doar doi parametri de intrare.
[16] Portul B este setat pentru a alimenta un motor prin pinii B0 .. B3 si celalat prin pinii B4 .. B7.

[17] Aceasta întârziere dintre fiecare pas este potrivită pentru motoare utilizate în unitățile de dischetă de 5-1/4 inch. Dacă întârzierea este prea mică, sau secvența de comandă nu este bună, motoarele vor vibra dar nu se vor învârti. Argumentul procedurii este de tip octet, deci trebuie sa fie într-un domeniu cuprins între 0 … 255.
[18] Două variabile sunt declarate pentru a ține valoarea inițială a pasului pentru fiecare motor. Valoarea inițială a ambelor este 0b_0001.
[24] Indicatoarele cu LED-uri sunt setate corespunzător cu valorile de intrare ale senzorilor.
[27] Fiecare motor este avansat cu o jumătate de pas numai când senzorul corespunzător se află în stare low. Procedura **stepper_motor_half_forward** este declarată în biblioteca jstepper, inclusă în biblioteca jlib. Procedura **stepper_motor_full_forward** poate fi deasemenea folosită. De observat ca Jal nu este sensibil la majuscule, procedurile pot fi scrise cu litere mari sau mici sau combinate.
[34] Procedura **steppers** este apelată pentru a genera noile valori bobinelor motoarelor şi pentru a aştepta intervalul de timp definit până la pasul următor.

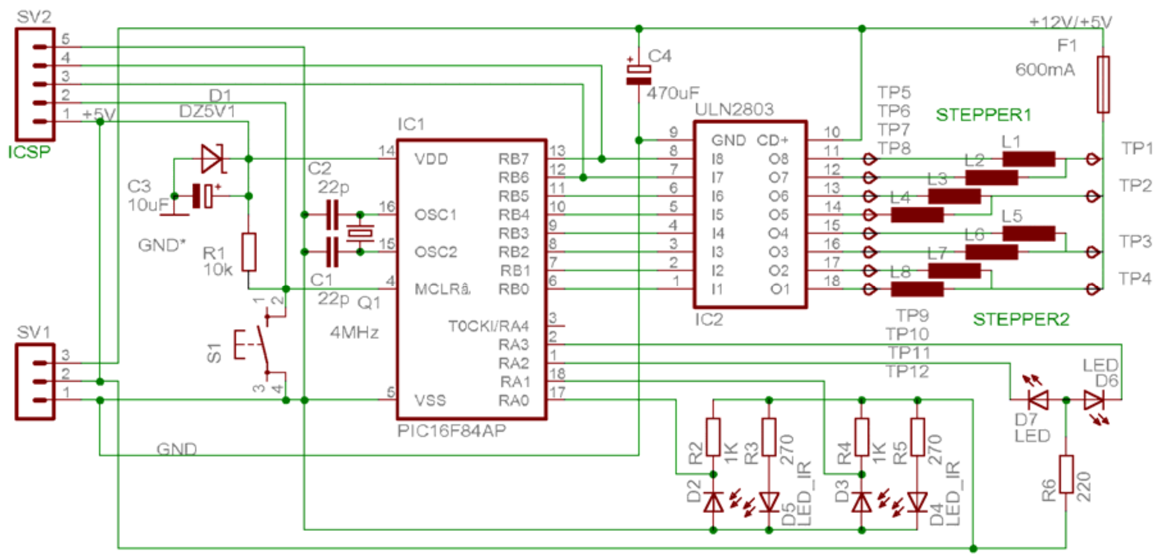


fig.2-2 Robot bimotor ce urmăreşte o linie trasată pe pardosea

In schema electronică se pot observa conectorul de alimentare SV1 (+5V, +12V) respectiv conectorul de programare în circuit SV2 al microcontrolerului. Driverul ULN2803 comandă direct bobinele motoarelor unipolare pas cu pas. Dacă sunt disponibile motoare cu tensiunea nominală de 5V atunci este necesară o singură tensiune de alimentare. O siguranță de 600mA, dimensionată corespunzător cu puterea motorului, protejează driverul împotriva supracurenților accidentali ce pot apare la blocarea mecanică a motorului sau în cazul comenzilor defectuoase. Condensatorul C4 trebuie montat în imediata vecinătate a motorului pentru a filtra spike-urile generate de acesta spre microcontroler. Optocuploarele reflective D2-D6 respectiv D3-D4 sunt de tipul celor utilizate în imprimante matriciale pentru detectarea capului de cursă (prin reflexie şi nu prin obturare!). Microelectronica Bucuresti produce un ansamblu detector IR cu reflexie cu fotodiodă si fototranzistor. Dacă se foloseşte acesta, fototranzistorii se conectează în locul fotodiodelor D2-D3 cu colectorii la RA0 şi RA1 iar emitorii la masă. LED-urile D6 şi D7 pot fi înlocuite cu o diodă LED bicoloră cu trei terminale. Dacă întreg ansamblul se alimentează din baterii, este importantă tensiunea de alimentare a microcontrolerului care e bine să nu depăşească datele de catalog,

motoarele pas cu pas pot fi subvoltate sau supravoltate fără probleme cu pînă la 20%. Astfel, montajul poate funcționa satisfăcător alimentat la 4.5V dacă motoarele pas cu pas au tensiunea nominală de 5V.

2.11.4 e0004 : afişarea temperaturii pe un display LCD

Programul următor citeşte temperatura de la un circuit integrat specializat, LM75, utilizând protocolul i2c, şi scrie rezultatul pe un afisaj LCD controlat de cipul Hitachi HD44780 sau compatibil.

```
[ 1]  -- afişarea temperaturii utilizând
[ 2]  -- un circuit LM75 şi un controler LCD cu
[ 3]  -- HD44780
[ 4]   include 16c84_10
[ 5]   include jlib
[ 6]   include lm75
[ 7]   include hd44780
[ 8]
[ 9]   const lm75_address = 0
[10]
[11]   hd44780_clear
[12]
[13]   forever loop
[14]
[15]      var byte t, d
[16]      var bit f
[17]
[18]      lm75_read_fdt( lm75_address, f, d, t )
[19]
[20]      hd44780_line1
[21]      if f then
[22]         hd44780 = "-"
[23]      else
[24]         hd44780 = "+"
[25]      end if
[26]      print_decimal_2( hd44780, d, " " )
[27]      hd44780 = "."
[28]      print_decimal_1( hd44780, t, "0" )
[29]
[30]      delay_200mS
[32]   end loop
```

[6,7] Bibliotecile **i2c** şi **hd447804** trebuie incluse în mod explicit deoarece nu fac parte din biblioteca jlib. Pentru a adapta pinii de intrare-ieşire corespunzători, utilizatorul trebuie să verifice bibliotecile **i2cp** şi **hd44780p.** Când schema electronică foloseşte alți pini decât cei mentionați în aceste biblioteci, utilizatorul trebuie să-şi modifice bibliotecile corespunzătoare şi să le includă în aplicația sa, în cazul de față conform schemei electronice din figură.
[9] Aceasta constantă defineşte pe trei biți adresa lui LM75 aşa cum este ea configurată de pinii A0, A1 şi A2 ai cipului. Rutinele lui LM75 vor seta biții mai semnificativi ai adresei (0b_1001).

[11] Afisajul LCD este şters.
[15,16] Sunt declarate variabilele pentru citirea lui LM75 .
[18] LM75 este citit.
[20] Cursorul afişajului este pus înapoi în prima poziţie. Utilizarea rutinei HD44780_clear în acest moment ar cauza o pâlpâire a afişajului.
[21]Este scris semnul. HD44780 este o pseudo-variabilă: fiecare asignare către această variabilă va chema o procedură care va scrie valoarea în afişaj şi va avansa cu o pozitie cursorul.
[26] Este afişată temperatura utilizând procedura **print_decimal_2**. Pseudovariabila HD44780 devine destinaţie pentru stringul format. Al doilea argument este valoarea ce urmeaza să fie scrisă. Ultimul argument este valoarea ASCII pentru zerourile mai semnificative (situate în stânga valorii afişate). In cazul de faţa va fi un spaţiu (blank)
[27,28] După punctul zecimal, este printată valoarea zecimală utilizând rutina **print_decimal_1** . Un "0" este generat aici, deci este printat fie "0" fie "5".
[30] O mică întârziere este inserată aici pentru a limita rata de afişare la o valoare convenabilă.

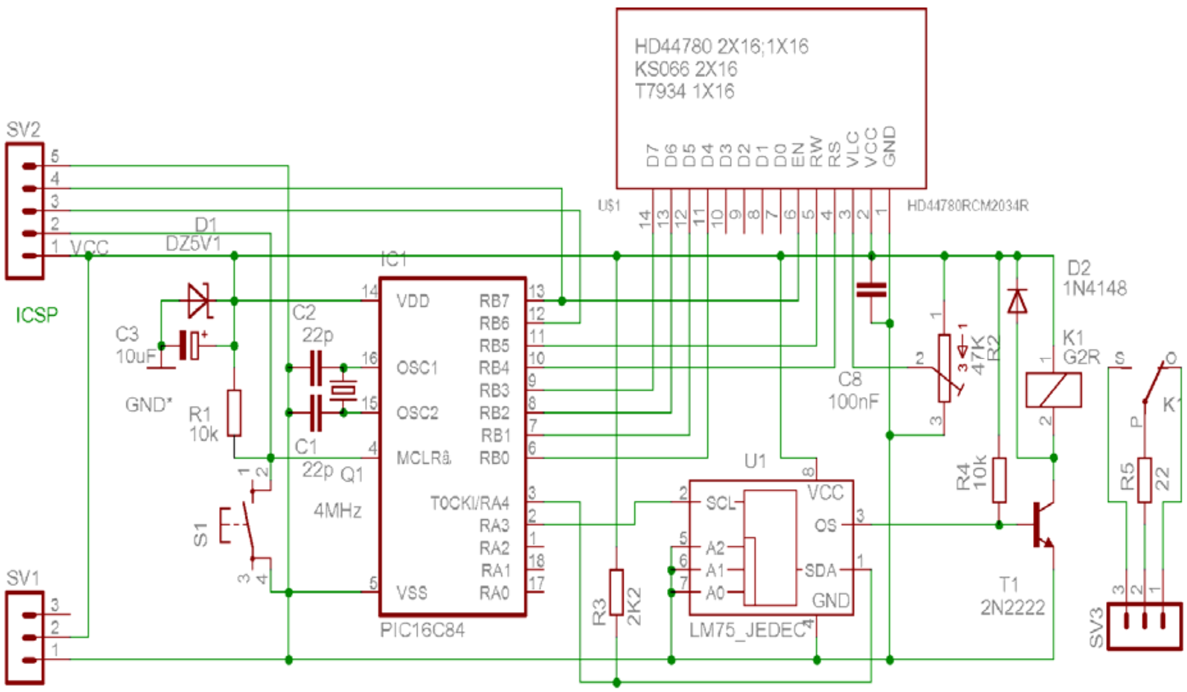


fig.2-3 Termometru/termostat inteligent cu LM75

Circuitul integrat LM75 (vezi foaia de catalog şi schema electronică) funcţionează atât ca şi termometru digital cât şi ca termostat. Deşi programul nu utilizează funcţia termostatului, ea este prezentă în schemă iar cititorul poate implementa ca un exerciţiu util, rutinele de programare a registrului intern ce actionează asupra ieşirii open drenă OS. Releul K1 are tensiunea nominală 5V şi curentul consumat este 30mA. Condensatorul C9 şi dioda D2 sunt amplasate fizic în imediata vecinătate a releului, împiedecând oscilaţiile parazite generate de acesta în momentul comutării, să perturbe funcţionarea microcontrolerului şi a circuitului LM75. Modul de conectare ai pinilor afişajului LCD şi ai termometrului LM75 trebuie specificaţi în bibliotecile HD44780p şi I2Cp sau într-o bibliotecă de definire a tuturor pinilor aparţinând componentelor implicate în proiect.

2.11.5 e0005: exemplu de utilizare a scrierii şi citirii din tabel şi eeprom

Programul următor exemplifică utilizarea memoriei program şi a memoriei eeprom pentru stocarea datelor.

```
include 16c84_10
include jlib
include hd447808
pragma eedata "H", "e", "l", "l", "o", " ", 0
procedure table is
pragma jump_table
  assembler
      addwf 2, f
      retlw "W"
      retlw "o"
      retlw "r"
      retlw "l"
      retlw "d"
      retlw 0
  end assembler
end procedure
procedure table_get( byte in x, byte out d )  is
  var byte s
  assembler
     bank  movfw x
           andlw 0x0F
     page  call table
     bank  movwf d
  end assembler
end procedure
procedure print_from_table is
  var byte d, i = 0
  forever loop
     table_get( i, d )
     if d == 0 then return end if
     hd44780 = d
     i= i + 1
     delay_100ms
  end loop
end procedure
procedure print_from_eeprom is
  var byte d, i = 0
  forever loop
     eeprom_get( i, d )
     if d == 0 then return end if
     hd44780 = d
     i= i + 1
     delay_100ms
  end loop
end procedure
forever loop
 hd44780_clear
 print_from_eeprom
```

```
[49]  print_from_table
[50]  delay_100ms( 5 )
[51] end loop
```

Liniile [1,3] conțin definirea bibliotecilor utilizate de program
[4] este modalitatea de memorare directă în eeprom prin pragma eedata, şirul se termină cu valoarea 0 pentru o uşoară identificare a sfârşitului acestuia; [5,16] procedură de memorare directă în memoria program prin adăugarea unui offset registrului PCL şi memorarea valorii ASCII în locații de memorie succesive. Limbajul JAL nu permite definirea precisă a primei locații de memorie unde se vor stoca datele, acest lucru este lăsat în seama compilatorului care o plasează în ultimul banc de memorie (numai variantele inițiale de compilator).
[17,25] procedură de citire a datelor din tabel, deoarece sunt doar 6 date memorate (incluzând terminatorul), se face o mascare [21] cu 0F pentru a reseta valoarea primilor 4 biți inutili, care pot avea orice valoare. Directivele *bank* şi *page* ce apar în limbajul de asamblare, sunt utilizate pentru schimbarea automată a bancului şi a paginii de memorie indiferent unde se găseşte porțiunea de cod. Este evident că în exemplul de față el va fi în acelaşi banc de memorie. Funcționarea procedurilor “print_from_table” [26-35] şi “print_from_eeprom” [36-45] sunt evidente, ambele utilizează un ciclu “forever loop” condiționat de identificarea terminatorului şirului (cifra zero). Procedurii de afişare HD44780 i se atribuie valoarea ce urmează a fi printată, după care se generează o întârziere pentru ca valoarea afişată să potă fi citită.
[46,51] este programul principal în care se apelează procedurile de citire din tabel şi eeprom, datele sunt afişate cu întârziere de 0.5 secunde, având ca efect curgerea mesajului pe display, caracter cu caracter, după care programul se reia.

2.12 Index rapid

noțiuni de bază

```
limbajul are un format liber, nu este sensibil la tipul de
caracter cu care se scrie, dar este necesară o pauză între
fiecare cuvânt editat
comentariile încep cu -- sau ; şi continuă până la sfârşitul
liniei
include jlib -- include biblioteca standard
```

tipuri

`bit`	:	1 bit
`byte`	:	8 bit, modulo-256
`universal`	:	numai în timpul compilării, maxim 32 biți întregi cu semn

literali

`bit`	:	true/high/on, false/low/off
`universal`	:	zecimal sau într-o altă bază (0b..., 0d..., 0h..., 0x...)
`byte`	:	"A" --valoarea ASCII

constante

`const bit  light   = low, blank = high`	-- constantă bit
`const byte pattern = 0b_0101_0101`	-- constantă octet

```
const ips           = clock / 2_500_000   -- constantă universală
```

variabile

```
var bit            done       =  false
var volatile byte status     at 3
var volatile bit  status_c   at status : 3
var                i2c_out    is pin_a3_direction
```

expresii matematice

```
prefix
   bit-bit          : !
   byte-byte        : + -
infix
   bit,bit-bit      : & | ^
   byte,byte-bit    :  = <= == !=
   byte,byte-byte   : & | ^ + - * / %
```

un universal poate fi folosit când este necesar un octet

instrucțiuni

```
var byte a a = 15 -- o declarație este considerată instrucțiune
if a  5 then ... end if
if a  0 then ... else ... end if
if a == 1 then ... elsif a == 2 then ... else ... end if
while ! done loop ... end loop
for 5 loop ... end loop
forever loop ... end loop
delay_1S( 5 )
asm clrwdt
assembler ... end assembler
```

declararea procedurilor şi a funcțiilor

```
procedure wait is for 100 loop asm nop end loop end procedure
procedure put( byte in x ) is ... end procedure
function get return byte is ... return x ... end function
```

pseudo-variabile

```
procedure x'put( byte in x ) is ... end procedure
function  x'get return byte is ... end function
```

Bibliografie:

1. Compilator JAL, http://www.voti.nl/jal/
2. Grup de lucru via email: http://groups.yahoo.com/group/jallist

Motto: *Prima dată le-am explicat şi n-au înțeles,*
A doua oară le-am explicat şi n-au înțeles,
A treia oară, explicându-le, aproape c-am înțeles şi eu…
dar ei tot n-au înțeles !

3 Interfațarea dispozitivelor periferice comune

Butoanele, tastatura, LED-uri simple, afişajul cu şapte segmente cu LED-uri, cu cristale lichide sau cu vacuum şi filament, relee, motoare pas cu pas, motoare de curent continuu şi cu reluctanță variabilă, encodere incrementale, difuzoare sau buzere (şi enumerarea ar putea să continue), sunt denumite generic dispozitive periferice. In acest capitol intenția autorului se îndreaptă spre modul de interfațare al acestora cu familia Microchip mid range, utilizând limbajul Jal şi assembler.

3.1 Primul program – un singur LED

După ce ne-am familiarizat cel puțin teoretic cu familia Microchip-flash, mid-range, e timpul să aplicăm ceva şi în practică. Dacă cel puțin exemplele "led care pulsează" şi "călăreț în noapte" au fost testate deja pe protoboard cu atât mai bine. Dacă nu, atunci este momentul de încă puțină teorie.

Puțini posesori ai unui calculator performant din ultima generație, cunosc că microprocesorul care este inima acestuia (şi care seamănă pe ici pe colo cu microcontrolerul nostru), nu poate face decât un singur lucru la un moment dat. Incredibil ! **Atunci cum este posibil să ruleze mai multe programe simultan, că doar aşa se vede pe ecran în cele cincizeci de ferestre Windows deschise în care lucrăm?** Acesta este efectul vitezei sporite a microprocesorului, a task managerului Windows-ului şi bineînțeles a acceleratorului grafic care dispune de procesor propriu şi memorie pe măsură. Utilizatorul de PIC-uri trebuie să memoreze această regulă de aur: PIC-ul nu ştie să facă decât un singur lucru odată ! Insă la fel ca PC–ul din exemplul nostru, poate realiza o sumedenie de lucruri în mod secvențial iar resursele hardware interne pe care acesta le are, pot executa anumite operații în paralel, comandate de firmware-le intern.

Un pin al microcontrolerului PIC16F628 (ales ca exemplu pentru că are un preț rezonabil) poate să fie intrare digitală, ieşire digitală, intrare analogică (de comparator) sau ieşire analogică (referință de tensiune sau ieşire de comparator), după secvența de program pe care o scrie utilizatorul. Starea PORTului A sau a PORTului B (sau a unui pin specific din aceste porturi) pentru modul IO (input-output, semnale digitale) este dată de regiştrii TRISx corespunzători. Pentru a seta direcția unui pin în JAL vom scrie:

```
include jpic628 -- biblioteci existente în distribuția pe CD
include jdelay
var volatile bit led is pin_b0
pin_b0_direction = output
```

Prezența bibliotecilor în care se defineşte cine este portul_b respectiv pin_b0, este deasemenea necesară (jpic628).

Variabila *led* odată definită, se poate scrie o buclă în care PIC-ul să execute ceva util :

```
for 10 loop led = on delay_1S ( 1 ) led = off
 delay_500mS ( 3 )
end loop
```

Liniile de program de mai sus, sunt echivalente din punct de vedere funcţional, cu diferenţe sinctactice minore, cu următorul ciclu:

```
for 10 loop
     pin_b0 = high
     delay_100mS ( 10 ) ; delay 100mS x 10 = 1S
     pin_b0 = low
     delay_100mS ( 15 ) ; delay 100mS x 15 = 1.5S
end loop
```

După ce este compilat, programul (denumit generic *blink.jal*) va face un led să pâlpâie, emiţând semnal luminos timp de cca. o secundă şi rămânând stins aproximativ o secundă şi jumătate, timp de zece ori, după care led-ul va rămâne stins până la o nouă resetare a microcontrolerului (prin deconectarea şi reconectarea alimentării sau scurtcircuitarea MCLR la masă).

Notă: *Imi pare rău că nu pot să-ţi vad faţa, cititorule, când reuşeşti să faci primul led să pâlpâie aşa cum doreşti tu şi nu cum vrea el, scriind un mic progrămel în Jal !*

- Este mai comod modul de definire al pinului care comandă led-ul, (variabila volatilă *led*) în detrimentul utilizării denumirii pinului conform bibliotecii jpic628 (pin_b0), mai ales dacă programul nostru final are peste 2000 de linii. Editorul PFE dispune de funcţia de înlocuire a unei expresii, însă folosirea de la bun început a declarării variabilelor după cum schema hardware şi logica dvs. o cere, simplifică înţelegerea ulterioră a programului, după ce praful s-a aşezat peste masa de lucru şi peste memoria utilizatorului.
- Analiza filei ***blink.asm*** rezultată din compilarea celor două variante de mai sus, va arăta că a doua variantă este ceva mai scurtă, deoarece definirea oricărei variabile suplimentare consumă din regiştrii SRAM ai microcontrolerului.
- Procedura ***delay_x (y)*** consumă timp preţios din timpul de lucru al PIC-ului, fiind un balast software pentru programul editat, timp în care microcontrolerul nu face nimic util ci se învârteşte într-una sau mai multe bucle de program, numărătorul de program şi alţi regiştrii auxiliari încrementându-se ***"rollover"*** (adică cu revenire la 0 după depăşirea valorii 255, sau modulo 256) până la obţinerea întârzierii dorite.

Un studiu ceva mai amănunţit al bibliotecii ***jpic628.jal*** şi a datelor de catalog ale microcontrolerului (CD:\datasheet\microchip\pic16f62xb), arată că în JAL nu se foloseşte explicit registrul TRISB pentru setarea direcţiei pinului b0 ci o pseudovariabilă numită ***trisb*** a cărei valoare se transferă apoi registrului fizic TRISB cu instrucţiunea ***tris 6*** (vezi cap1.5.6). Acest lucru înseamnă că mai avem o posibilitate de a defini direcţia variabilei *led* într-o altă formă decât cea cunoscută:

```
var volatile byte hw_trisb at 0x_86 -- vezi fig.1-15
bank_1
hw_trisb = 0b_1111_1110
```

```
        -- toţi pinii portului b sunt intrări (1) mai puţin b0 care este ieşire(0)
bank_0
```

Schimbarea denumirii registrului *trisb* în *hw_trisb* a fost necesară pentru a evita conflictul de definire a mai multor variabile cu acelaşi nume. Nu uitaţi să schimbaţi bancul de lucru de fiecare dată când registrul asupra căruia operaţi se găseşte altundeva decât în bancul 0 unde compilatorul (şi utilizatorul) se simte în apele lui.

3.2 Acelaşi LED şi ceva mai mult…

In câte feluri se poate interfaţa un LED la pinii unui microcontroler ? Un LED monocolor sau bicolor cu conectarea ambazelor în antiparalel, având doar două terminale, nu poate fi conectat decât în exact trei moduri:

- între pinul PIC şi masă , comanda pe anod
- între pinul PIC şi Vcc, comanda pe catod
- între doi pini ai PIC-ului, comanda pe anod şi pe catod

Nu mai trebuie să menţionăm că LED-ul are ca parametru de comandă curentul şi deci necesită limitarea acestuia la valoarea maximă acceptată LED şi/sau generată de microcontroler (20mA dinspre Vcc, 25mA spre masă, valori maxime absolute pe pinul PIC-ului), căderea de tensiune pe joncţiune fiind un parametru secundar care trebuie luat în calcul la dimensionarea rezistenţei de balast, mai ales când se utilizează LED-uri conectate în serie sau LED-uri albastre care au o cădere mare de tensiune pe jocţiuni. Un efect interesant poate fi obţinut în exemplul următor:

```
include 16f84_4
include jpic
include jdelay

var bit led is pin_b0      ; un LED monocolor
pin_b0_direction = output
var bit buton is pin_b7    ; un pushbutton simplu
pin_b7_direction = input
var byte x = 20

  while x > 2 loop
     if buton == low then          -- dacă s-a apăsat butonul
        x = x - 1                  -- decrementează variabila
        led = on                   -- aprinde led-ul
         delay_10ms( x )           -- şi ţine-l aprins t = 10mS * variabila
         led = off                 -- stinge led-ul
        delay_10ms( x )            -- şi ţine-l stins t = 10mS * variabila
     else                          -- dacă s-a ridicat  butonul
         x = 20                    -- setează variabila la valoarea iniţială
         led = off                 -- asigură-te ca led-ul e stins
     end if                        -- şi opreşte-te
  end loop
```

Programul anterior face citirea rudimentară a unui buton conectat pe pinul b7 faţă de masă (rezistenţă de pull-up de 10K din acelaşi punct la Vcc) şi realizează o pâlpâire cu durată

variabil-descrescătoare a LED-ului conectat pe pinul b0 față de masă, proporțională cu timpul de apăsare al butonului. Variațiuni pe aceeaşi temă se pot obține foarte simplu dacă se face o incrementare a variabilei x (utilizând câteva instrucțiuni în limbaj de asamblare doar pentru a evidenția că există şi această posibilitate)

```
; option = 0b_0xxx_xxxx
var byte x = 1
  forever loop
    while x < 50 loop
     if ! pin_b7 then
       asm incf x       ; x = x + 1
       asm bsf pin_b0   ; instrucțiune în limbaj de asamblare, bit set f
       delay_10ms( x )
       asm bcf pin_b0   ; idem bit clear f
       delay_10ms( x )  ; delay variabil
     else
        x = 1
        led = off
     end if             ; end buclă if
    end loop            ; end buclă while
  end loop              ; end buclă forever
```

Cu acelaşi buton şi LED se poate implementa foarte simplu un cifru electronic de 8 biți. De această dată LED-ul este conectat cu anodul la Vcc şi cu catodul la pin_a0 printr-o rezistență de limitare a curentului de 330 ohmi. Butonul utilizează aceeaşi rezistență de pull-up de 10K. Nu sunt figurate elementele comune necesare funcționării microcontrolerului şi anume: oscilatorul cu cuarț împreună cu cele două condensatoare de 15…33pF conectate la masă de pe pinii osc_in şi osc_out, respectiv rezistența ce asigură nivelul logic high pe MCLR, şi condensatorul de filtraj al alimentării microcontrolerului. Aceste elemente vor fi **omise în mod deliberat** din următoarele scheme prezentate, pentru a focaliza ochii cititorului pe aplicația descrisă în program.

```
include 16f84_10
include jlib

pin_a0_direction = output         -- led şi/sau releu
pin_a1_direction = input          -- buton
port_b_direction = all_output
procedure generare_cod ( byte out x ) is
   for 8 loop                     -- repetă de 8 ori
      var byte n = 0
      while pin_a1 == high loop   -- când nu se apasă pe buton
         delay_1mS( 5 )           -- întârziere de 5 mS
         n = n + 1                -- incrementarea variabilei
         if n == 255 then         -- rollover ?
            x = 0                 -- da, returnare x = 0, nu s-a apăsat butonul
            return
         end if
      end loop
      delay_1ms( 200 )            -- delay 200mS
      x = x << 1                  -- rotire o poziție spre stânga
```

```
      if pin_a1 == low then          -- s-a apăsat butonul odată ?
         x = x + 1                   -- incrementează variabila de ieşire
      end if
      while pin_a1 == low loop end loop
                                     -- cât timp butonul stă apăsat, nu execută nimic
   end loop                          -- am terminat cele 8 cicluri
end procedure
```

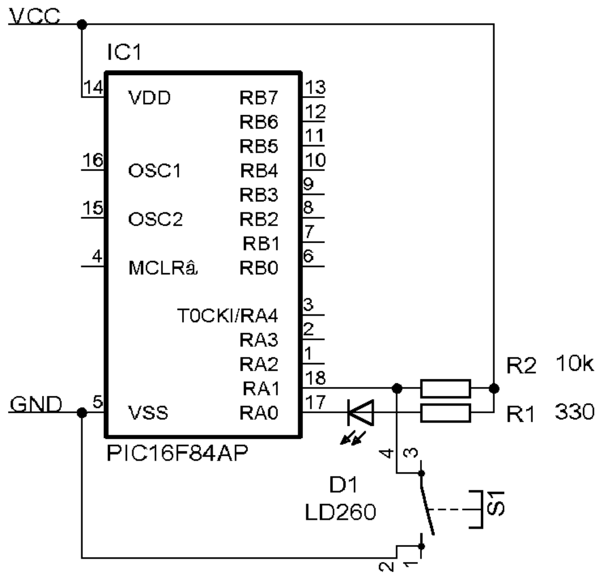


Fig.3-1 Cifru electronic cu un led şi un buton

```
var byte code
forever loop
   pin_a0 = high                    -- LED-ul stins
   if pin_a1 == low then            -- butonul apăsat ?
      generare_cod( code )          -- da, generează codul
      if code == 0b_0000_1111 then          -- cod echivalent cu cel prescris ?
                                    -- codul este 4 apăsări scurte + 4 apăsări lungi
         pin_a0 = low                       -- da, aprinde led-ul
         delay_1s( 5 )                      -- timp de 5 S
      else                                  -- cod incorect ?
         delay_1s( 10 )                     -- aşteptă 10 S şi introdu-l din nou
      end if
   end if
end loop                                    -- reia ciclul
```

In exemplele anterioare am fost nevoiţi să utilizăm o rezistenţă de "pull-up" a pinului de intrare pe care era conectat butonul. O eroare hardware care putea fi evitată dacă citeam cu mai mult interes foaia de catalog a microcontrolerului referitoare la regiştrii cu funcţii speciale şi anume registrul ***option.*** Aşa cum se observă în fig.3-2, registrul ***option*** dispune de bitul /RBPU (bit7) care, dacă este în starea *low,* conectează în interiorul microcontrolerului o rezistenţă de pull-up (de fapt e un tranzistor MOS, conexiunea drenă-sursă) ce asigură un curent maxim de 400uA în fiecare pin de intrare al portului B. Setând registrul ***option*** corespunzător, înainte de a utiliza pinul de intrare respectiv, nu mai este nevoie de rezistenţă exterioară:

```
option = 0b_0xxx_xxxx
```

Studiind biblioteca jpic628, se observă din nou că scrierea în registrul option se face printr-o pseudo-variabilă (cap.2.6.3.) şi nu în mod direct prin acces la registrul fizic ***option.*** Această pseudovariabilă este definită în biblioteca jpic:

```
var volatile byte option
procedure option'put( byte in x ) is
   assembler
      bank  movfw x
            option
   end assembler
end procedure
```

Rolul acestui registru este ceva mai important dacă se lucrează cu timerul ***tmr0*** (timerul de bază în microcontrolerele cu 18 pini), aşa cum vom vedea în continuare.

RBPU	**INTEDG**	**TOCS**	**TOSE**	**PSA**	**PS2**	**PS1**	**PS0**
7R/W	6 R/W	5 R/W	4 R/W	3 R/W	2 R/W	1 R/W	0 R/W

RPBU: bitul de setare al pull-up portB
1 = pull-up portB este dezactivat
0 = pull-up portB este activat de valorile individuale ale latchurilor (numai pe intrări)

INTEDG: bitul de selecție al frontului intreruperii RB0/INT
1 = intrerupere pe front crescător RB0/INT
0 = intrerupere pe front descrescător RB0/INT

T0CS: selecția tactului pentru TMR0
1 = tactul este semnalul provenit din RA4/T0CKI
0 = tactul intern

T0SE: selecția polarității semnalului de tact extern
1 = incrementare pe tranziția low-high a RA4/T0CKI
0 = incrementare pe tranziția high-low a RA4/T0CKI

PSA: bitul de asignare al prescalerului
1 = prescaler utilizat de WDT
0 = prescaler utilizat de TMR0

PS2:PS0: biții de setare ai ratei de divizare în prescaler

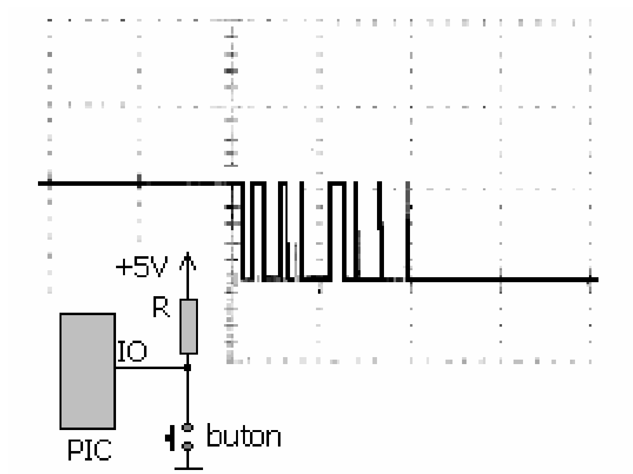
	TMR0	WDT
000 =	1:2	1:1
001 =	1:4	1:2
010 =	1:8	1:4
011 =	1:16	1:8
100 =	1:32	1:16
101 =	1:64	1:32
110 =	1:128	1:64
111 =	1:256	1:128

Fig.3-2 Registrul OPTION

Semnificația notației R/W este citeşte (Read)-scrie (Write).

3.3 Butoane şi matrici de butoane

In exemplul precedent, prima citire a butonului nu este foarte corectă datorită fenomenului de comutaţie parazită care apare atât la închiderea cît şi la deschiderea unui *push*-buton simplu. In funcţie de tipul de buton, durata necesară ca semnalul să se stabilizeze este cuprins între 2 şi 8 mS, uneori chiar mai lung. De aceea modul cel mai simplu de a obţine o citire corectă a stării butonului este interogarea repetată a intrării de cel puţin două ori, la intervale de timp mai mari decât timpul de stabilizare.



In imagine se observă efectul comutării intrării din starea *high* în stare logică *low*, diviziunea verticală fiind de 5V iar cea orizontală de 1mS.
Inevitabil, observăm că evenimentele care trebuiesc identificate şi tratate cu ajutorul microcontrolerului sunt strict dependente de timp. Una din posibilităţi este următoarea:

fig.3-3 Comutarea parazită la acţionarea unui buton

```
var bit stare = low
procedure buton read ( bit out stare ) is
    if ! buton then delay_1mS ( 10 )
        if ! buton then
             stare = high
         end if
    else stare = low end if
end procedure
```

Procedura simplă de citire a stării butonului utilizează din nou o întârziere software de 10mS, se verifică dacă starea butonului este *low*, dacă da, se asigură o întârziere suficient de mare ca butonul să reuşească să treacă în starea *low* permanent şi se face o nouă citire a stării butonului. Dacă butonul a rămas în *low* atunci se execută codul utilizatorului (aici se setează variabila *stare*). Bineînţeles că variabila *stare* trebuie definită şi resetată în prealabil pentru ca programul să funcţioneze corect. Există şi o problemă de dinamică a citirii butonului; este posibil (dacă nu se asigură un interval suficient de mare între două citiri), ca o singură apăsare de buton să dureze mai mult de 10mS şi să fie interpretată ca apăsare succesivă. Atunci, o singură apăsare de buton va genera în realitate două (sau mai multe) comenzi repetate (variabila *stare* este resetată după fiecare acţionare de buton în programul principal, *main loop*). Acest lucru poate fi benefic pentru incrementarea continuă a unui registru atât timp cât butonul este menţinut apăsat, sau poate fi deranjant dacă se urmăreşte realizarea unei singure comenzi, indiferent de durata de apăsare a butonului respectiv.

O metodă mult mai inteligentă este utilizarea unuia dintre temporizatoarele interne ale PIC-ului pentru generarea intervalelor precise de timp necesare pentru întârzieri. TMR0 este un registru de 8 biți care are următoarele facilități:

- Poate fi utilizat ca *timer* (temporizator) sau *counter* (numărător)
- Dispune de un *prescaler* de 8 biți (un numărător de 8 biți)
- Tactul este selectabil intern sau extern via pin_a4, fronturile de comutare sunt selectabile
- Registrul TMR0 poate fi citit şi scris de către utilizator
- Este generată o întrerupere la depăşirea valorii 0xFF a registrului TMR0

Prescalerul este utilizat "la comun" de TMR0 şi de WDT. In esență *prescalerul* memorează de câte ori va fi incrementat TMR0 până la depăşire *(rollover*). Incrementarea registrului TMR0 porneşte de la valoarea scrisă de utilizator în el. Orice modificare a lui TMR0 va inițializa şi *prescalerul*. Cum se setează acesta ca accesoriu al TMR0 se poate urmări în fig.3-2 şi în procedura următoare:

```
procedure tmr0_rollover is
-- ------------------------------------------------
clear_watchdog   -- initializează timer0/prescaler for x_ms rollover
option = 0b_0000_0111 - fig.3-1
-- set prescaler at 7:1/256
--                 6:1/128
--                 5:1/64
--                 4:1/32
--                 3:1/16
--                 2:1/8
--                 1:1/4
--                 0:1/2
-- exemplu: t = 65 ms, Fclk = 4000, option = 7, tmr0 = 0
tmr0 = 0
end procedure
```

Ecuația care guvernează funcționarea TMR0 este:

$$tmr0 = 256 - \frac{t \times Fclk}{4 \times prescaler}$$

Unde: *prescaler* este valoarea prescaler-ului (1, 2, 4, 8, 16, 32, 64, 128, 256)
tmr0 este valoarea necesară (0…255) pentru a fi înscrisă în registru TMR0
t este durata de timp dorită (în mS)
Fclk este frecvența oscilatorului (în kHz)

respectiv:

$$t_{real} = \frac{4 \times prescaler \times (256 - tmr0)}{Fclk}$$

Datorită operării cu constante întregi, se observă că pentru valori uzuale ale frecvenței de tact, nu se pot obține decât valori de timp apropiate de cele dorite. De exemplu pentru un timp necesar t = 20mS la o frecvență oscilator de 4MHz şi un *prescaler* 1:256, rezultatul pentru $tmr0_{calculat}$ = 178 este t_{real} = 19.96mS. Pentru o întârziere necesară citirii unui buton, valoarea obținută este corespunzătoare, dar dacă aceeaşi valoare este folosită pentru generarea unui orologiu de timp real de 1S, eroarea la fiecare secundă va fi de 2mS (50x 19.96mS = 998mS), suficient de mare ca să necesite corecție software. Nu acelaşi lucru se întâmplă dacă utilizăm un cuarț de 32.768 KHz (2^{15} Hz), rezultatul va fi obținerea unei cuante întregi de timp. Momentul când TMR0 execută un *rollover* este semnalizat prin setarea bitului *t0if* din registrul *intcon* (fig.3-4), bit ce trebuie resetat prin software după ce este citit, pentru a permite o nouă detectare a evenimentului. Observați că nu e nevoie de întreruperi pentru a citi/modifica valoarea acestui bit, deci momentan faceți abstracție de existența întreruperilor tratându-le ca inexistente.

Vom complica exemplul de citire a butonului nostru adăugând încă unul pentru a diversifica aplicația inițială (bibliotecile incluse şi fila de definire a PIC-ului nu vor mai apare în exemple, însă cititorul a învățat deja că existența lor este necesară la compilarea fiecărui program). Butonul *but1* (pin B6, activ *low*), este citit folosind un algoritm software fără *delay* conținut în liniile comentate cu *1,*2,*3 ale programului următor. TMR0 este folosit doar pentru a genera o întârziere de aproximativ 1.3 S necesară aprinderii LED-ului, acesta fiind efectul vizibil al apăsării butonului *but1*. Dacă linia *3 lipseşte, este activă doar prima apăsare a butonului *but1*, resetarea variabilei *curent_push* fiind singura care activează butonul pentru următoarea apăsare. Plasarea corectă a momentului acestei resetări permite activarea sau dezactivarea funcției butonului cu o întârziere generată doar de curgerea normală a programului. Citirea butonului *but2* (pin B7, activ *low*) se face însă la intervale de timp egale, definite de TMR0 (în exemplu la 65mS). Efectul apăsării pe *but2* este setarea flagului *flag*, cu rol de condiție pentru pâlpâirea LED-ului de zece ori cu raportul aprins/stins de 200mS/300mS. Dacă într-un program complex, butoanele sunt citite în programul principal (definit de bucla *forever loop…end loop)* şi execuția rutinei care citeşte butoanele este plasată tot acolo (ca în cazul lui *but1*), rezultatul va fi obținerea unor întârzieri varibile la citirea butoanelor, dependente timpul de execuție al porțiunii de cod ce va fi executată între setarea şi resetarea flagului corespunzător butonului. De aceea, dacă utilizarea butoanelor se face înafara întreruperilor, este importantă păstrarea aproximativ constantă a timpului de execuție al acestor porțiuni de cod pentru toate butoanele implicate.

```
tmr0_rollover          -- utilizăm procedura anterioară fără parametri, cu rollover la 65mS
pin_b6_direction = input            -- intrări pentru butoane
pin_b7_direction = input
var volatile bit next_push          -- variabilă "specială"
var bit led is pin_b0
pin_b0_direction = output
led = off                           -- ne asigurăm ca LED-ul e stins
var byte counter = 0                -- registru numărător
var bit flag = low                  -- un simplu flag auxiliar

forever loop
next_push = ! pin_b6                -- but1 = ( next_push ^ curent_push ) & next_push
if ( next_push ^ curent_push ) & next_push then        -- *1
      curent_push = next_push                          -- *2
        if intcon_t0if then
```

```
          counter = counter + 1 intcon_t0if = low
        end if
          if counter < 20 then led = on    -- 20 x 65mS = 1300mS
          else counter = 0 led = off
          end if
       curent_push = low                   --*3
 elsif (! pin_b7) & intcon_t0if then       -- but2 = (! pin_b7) & intcon_t0if
       flag = on intcon_t0if = low
 end if
if flag then
 for 10 loop led = on  delay_10mS( 20 )
             led = off delay_10mS( 30 )
 end loop
 flag = off  --  resetăm flagul pentru următoarea apăsare a but2
end loop
```

GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
7R/W	6R/W	5 R/W	4 R/W	3 R/W	2 R/W	1 R/W	0 R/W
GIE: bitul de selecţie al întreruperilor globale 1 = activează întreruperile globale nemascate 0 = dezactivează toate întreruperile							
PEIE: bitul de selecţie al întreruperilor periferice 1 = activează întreruperile periferice 0 = dezactivează toate întreruperile periferice							
T0IE: bitul de setare al semnalizării depăşirii TMR0 1 = activează semnalizarea depăşirii 0 = dezactivează semnalizarea depăşirii							
INTE: bitul de setare al întreruperii externe RB0/INT 1 = activează întreruperea RB0/INT 0 = dezactivează întreruperea RB0/INT							
RBIE: bitul de setare al intreruperii la schimbarea stării logice a portului B (intrare) 1 = activează întreruperea 0 = dezactivează întreruperea							
T0IF: bitul de semnalizare al depăşirii valorii maxime (255) pentru TMR0 1 = valoarea maximă pentru TMR0 s-a depăşit, resetarea software a acestui bit este obligatorie 0 = nu s-a depăşit valoarea maximă a TMR0							
INTF: flag-ul de întrerupere al RB0/INT 1 = a avut loc întreruperea pe RB0/INT (necesită resetare software) 0 = nu a vut loc întrerupere pe RB0/INT							
RBIF: flagul de întrerupere la schimbarea stării portului B 1 = cel puţin una din întrările RB7:RB4 şi-a schimbat starea logică, lipsa stimulului va continua să menţină bitul setat, citirea întregului port B va permite ştergerea bitului corespunzător întreruperii; RBIF trebuie resetat prin software 0 = nici un pin RB7:RB4 nu şi-a schimbat starea INTCON							

fig. 3-4 Registrul INTCON

Modul de citire al butonului *but2* este analizat grafic în continuare (fig.3-5).

Citirea stării *but2* se face o singură dată la intervale egale de timp definite de trecerea lui *intcon_t0if* în stare *high.* Privind tranziţia zgomotoasă a semnalului din stare

high în stare *low* (fig.3-5) observăm că citirea butonului se face asincron, relativ la fronturile parazite de comutaţie, deci este posibil ca o scurtă apăsare pe buton să nu fie detectată corect de către acest program. Corectarea problemei este simplă, şi are la bază citirea stării *but2* de două sau de mai multe ori, la intervale de timp suficient de mari pentru eliminarea comutării parazite (faza de confirmare din fig.3-5).

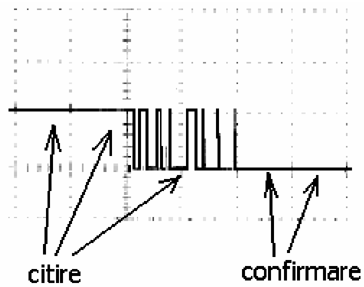


fig.3-5 Validarea citirii butonului

```
var bit control_2 = low
var bit but2 = low
 if (! pin_b1 & tmr1if) then
    control_2 = on tmr1if = low
 end if
 if (control_2 & tmr1if ) then
         but2 = on control_2 = off tmr1if = off
 end if
```

După executarea codului utilizator corespunzător apăsării pe buton, în main loop, este necesară resetarea variabilei *but2*, pentru o nouă citire. Se poate remarca o noutate în programul anterior. S-a utilizat TMR1 în locul lui TMR0 şi testarea bitului *tmr1if* corespunzător registrului PIR1 (cap.5 fig.5-3). Bineînţeles că este necesară iniţializarea prealabilă a TMR1 şi pornirea acestuia:

```
assembler
  bcf tmr1cs        -- tmr1 în timer mode, internal clk, fosc/4
  bsf t1ckps1
  bcf t1ckps0       -- prescaler 1:4
  bcf t1oscen       -- stop oscilator extern
  bcf tmr1on        -- tmr1 este oprit
  bcf intcon_gie    -- toate întreruperile dezactivate
  bsf status, 5
  bcf status, 6     -- bank_1
  bcf tmr1ie        -- overflow interrupt dezactivat
  bcf status, 5     -- bank_0
  bcf status, 6
end assembler
  tmr1l = 64
  tmr1h = 64        -- perioada de overflow
  asm bsf tmr1on    -- start tmr1
```

Suntem în nefericita ipostază de a avea două necunoscute: din nou întreruperile (puţintică răbdare !) şi o nouă resursă internă, timerul TMR1. Diferenţa semnificativă între TMR1 şi TMR0 este că având 16 biţi poate soluţiona întârzieri mai mari decât TMR0, rezoluţia fiind asigurată de regiştrii *tmr1l* şi *tmr1h,* nu mai este nevoie de un *prescaler* aşa de precis ca în

cazul lui TMR0, doar doi biţi fiind suficienţi (patru valori zecimale: 1, 2, 4, 8). TMR1 poate funcţiona cu tact intern sau extern, în mod *timer* sau numărător sincron/asincron. Regiştrii asociaţi acestuia (în interiorul cărora se produc evenimente legate de TMR1) sunt: PIR1 (fig.3-15), PIE1, TMR1L, TMR1H (cei doi regiştrii de 8 biţi ai TMR1) şi T1CON fig.3-6, (CD:\ datasheet\microchip\)
Pentru exemplul de mai sus:

tmr1l = 64 = 0x40 tmr1h = 64 = 0x40

$$t = TMR1 \times prescaler \times Tcy$$

prescaler = 1:4
Tcy = 1/(4000000/4) Hz = 1μS. Oscilatorul de 4MHz este divizat intern cu 4.
Valoarea lui TMR1 de la care începe incrementarea şi până la setarea flagului *tmr1if* este : 0x4040 (hexa) adică 16448 (zecimal). Durata totală va fi :

TMR1 = 256 x TMR1H + TMR1L (valori zecimale ale tuturor regiştrilor)

adică: *t* = 4x16448x1uS = 65792uS = 65.8mS . Observaţi că din punct de vedere al compilatorului nu există diferenţe între majuscule şi minuscule, tmr1l şi TMR1L fiind echivalente.

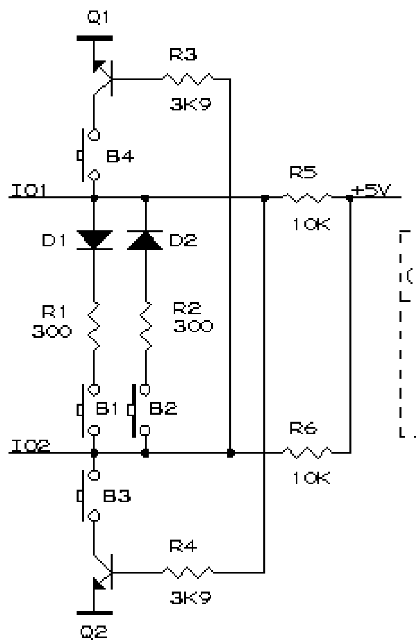
-	**-**	**T1CKPS1**	**T1CKPS0**	**T1OSCEN**	**T1SYNC**	**TMR1CS**	**TMR1ON**
7	6	5 R/W	4 R/W	3 R/W	2 R/W	1 R/W	0 R/W
T1CKPS1:T1CKPS0: biţii de selecţie ai prescalerului pentru TMR1 11 = 1:8 10 = 1:4 01 = 1:2 00 = 1:1							
T1OSCEN: bitul de control al oscilatorului TMR1 1 = oscilatorul este pornit 0 = oscilatorul este oprit, mod de consum redus							
T1SYNC: bitul de control al sincronizării exterioare pentru oscilatorul TMR1 Pentru TMR1CS = 1 1 = nu sincronizează tactul extern 0 = sincronizează tactul extern Pentru TMR1CS = 0 bitul este ignorat, TMR1 foloseşte tactul intern							
TMR1CS: bitul de selcţie al sursei de tact pentru TMR1 1 = tact extern din pinul RC0/T1OSO/T1CKI pe front crescător 0 = tact intern egal cu fosc/4							
TMR1ON: bitul de start al TMR1 1 = porneşte TMR1 0 = opreşte TMR1							T1CON

fig.3-6 Registrul T1CON

Una din problemele pe care le ridică utilizarea a **n** butoane, este consumarea resurselor hardware a PIC-ului dacă se interfaţează un singur buton pe un pin IO. Există mai multe moduri distincte de a micşora numărul pinilor necesari pentru interfaţarea a **n** butoane după cum urmează:

3.3.1 Interfațarea a 4 butoane pe 2 pini de intrare-ieşire

In mod normal, pentru decodarea a $\mathbf{2^n}$ stări logice sunt necesari **n** pini de intrare, conform principiului de conversie din sistemul binar (microcontroler) în cel zecimal (butoane on-off), sau dacă se consideră ca stare distinctă şi starea de înaltă impedanță, atunci $\mathbf{3^n}$ stări logice pot fi decodate cu **n** pini de intrare. Prima variantă permite o interfațare economică, salvând un număr important de pini pentru alte comezi, situație benefică în cazul microcontrolelor cu număr redus de pini. Schema următoare evidențiază o modalitate de interfațare utilizând doar câteva componente discrete.



IO1 (Rb2)	IO2 (Rb3)	BUTTON
IN	LOW	B1
IN	HIGH	B4
LOW	IN	B2
HI	IN	B3
LOW	HI	X
HI	LOW	X

fig.3-7 Patru butoane pe doi pini IO

Explicarea funcționării este simplă dacă cititorul îşi reaminteşte corolarul formulat la începutul capitolului: PIC-ul efectuează o singură operațiune la momentul *t*, dar poate executa *n* operațiuni în mod secvențial. Pentru citirea butonului B1, IO1 este intrare iar IO2 este ieşire în starea logică *low.* Acționând B1, intrarea IO1 trece din starea logică *high* menținută de R5, în stare *low*, potențialul pinului fiind menținut aproximativ la valoarea (R1/R1+R5)Vcc. Analog, pentru acționarea lui B2, IO1 devine *low* iar IO2 devine intrare. Diodele D1, D2, R1, R2 au rolul de a proteja pinii IO1 şi IO2, dacă aceştia nu sunt programați cu secvența corespunzătoare şi ambele butoane B1 şi B2 sunt apăsate simultan. Pentru citirea butoanelor B3 şi B4 au fost necesare două inversoare cu tranzistoare. De exemplu, citirea butonului B4 implică IO1 să fie intrare iar IO2 să fie ieşire în starea logică *high*. IO2 fiind high, tranzistorul Q1 este polarizat direct (se găseşte în conducție) şi apăsarea butonului B4 va trece intrarea IO1 din stare logică *high* în stare logică *low.*

Programul care citeşte cele patru butoane, interfațate pe pinii Rb2 respectiv Rb3 şi asigură 7 funcții distincte pentru cele patru butoane, este prezentat în continuare:

```
var bit button_flag = low
forever loop
            -- citeşte butonul 1 (butonul 1 schimbă funcția  butoanelor 2 , 3 şi 4)
pin_b3_direction = output
pin_b3 = low
pin_b2_direction = input
 if ! pin_b2  then
  button_flag = ! button_flag
 end if
```

```
pin_b3 = high                                -- citeşte butonul 4
 if ( ! pin_b2 ) & button_flag then          -- execută funcția 1 a acestui buton
 elsif ( ! pin_b2 ) & ( ! button_flag ) then
                                              -- execută funcția 2 a acestui buton
 end if
pin_b2_direction = output                     -- citeşte butonul 2
pin_b2 = low
pin_b3_direction = input
 if ( ! pin_b3 ) &  button_flag  then         -- execută funcția 1 a acestui buton
 elsif ( ! pin_b3 ) & ( ! button_flag ) then
                                              -- execută funcția 2 a acestui buton
 end if
pin_b2 = high
if ( ! pin_b3 ) & button_flag then            -- citeşte butonul 3
                                              -- execută funcția 1 a acestui buton
 elsif ( ! pin_b3 ) & ( ! button_flag ) then
                                              -- execută funcția 2 a acestui buton
 end if
end loop
```

In mod evident, programul prezentat nu dispune de nici o tehnică de eliminare a comutațiilor parazite a butoanelor, pe baza unor întârzieri generate software sau hardware prin TMR0 sau TMR1. Metodele prezentate anterior trebuiesc aplicate în programul funcțional de mai sus.

3.3.2 Taste funcționale – o privire de ansamblu

Indiferent de modul de conectare al butoanelor la microcontroler, metoda care utilizează mai multe butoane pentru un număr de comenzi distincte ce depăşeşte numărul butoanelor, poartă denumirea generică de taste funcționale. Cel mai concludent exemplu (dar şi cel mai prost mod de implementare) este telefonul celular utilizat pentru mesagerie scurtă (SMS). Un număr redus de taste permite scrierea întregului alfabet, a caracterelor numerice şi a semnelor iar manevrarea meniurilor se face de obicei din trei butoane. Metoda software care a fost utilizată în exemplul următor, se numeşte metoda semaforului de meniu şi îl are pe autor drept naş. Vom numi cele trei valori ale semaforului din exemplu, în mod identic cu realitatea întâlnită în circulație de către pieton:

```
const byte rosu  = 1
const byte galben = 2
const byte verde = 0
var volatile byte semafor
var bit int_t0if = low
var bit but1, but2, but3, but4
no_ad ; dezactivează intrările analogice a0…a3 (vezi biblioteca janalog.jal) pt. funcționare digitală
```

Cele patru butoane (fig.3-8) se găsesc conectate față de masă pe intrările a0, a1, a2 şi a3, ale unui PIC16F877, citirea acestora se face într-o rutină de întreruperi (vezi cap. 5), utilizând TMR0 setat pentru un rollover de 65 mS, intervalul de timp este multiplicat cu 2 pentru o citire corectă la schimbarea meniului.

```
procedure interrupt_service_routine is
pragma interrupt
if intcon_t0if then kbd = kbd + 1 – întârziere pentru citirea  butoanelor
  if kbd == 2 then                  -- 2x65ms = 130ms
     int_t0if = intcon_t0if
      kbd = 0
  end if
end if
 if    (! pin_a0) & int_t0if then but1 = on
 elsif (! pin_a1) & int_t0if then but2 = on
 elsif (! pin_a2) & int_t0if then but3 = on
 elsif (! pin_a3) & int_t0if then but4 = on
 end if
end procedure
```

-- programul principal testează starea semaforului şi în cadrul ramurii de program corespunzătoare, execută citirea variabilelor setate de către butoanele în cauză cât şi alte rutine utilizator, schimbând corespunzător valoarea semaforului
-- proceduri predefinite cu rol nesemnificativ în acest exemplu şi care nu au fost explicitate sunt:

```
display_tmp                         ;  afişarea simbolului timp
display_pwr                         ;  afişarea simbolului putere
display_pwr_value                   ;  afişarea valorii puterii
display_grafic                      ;  afişarea unor simboluri grafice predefinite
display_tratament                   ;  afişarea meniului de tratament
display_stby_off                    ;  meniul stand-by/off
eeprom_read_877 ( address, data ) ;  rutina de citire a eeprom-ului  intern
display_minute                      ;  afişarea minutelor
eeprom_write_877 ( address, data )  ;  rutina de scriere în eepromul intern
display_first                       ;  meniul inţial
pin_d1_direction = output           ;  un buzzer se găseşte conectat pe pinul d1 - masă
procedure tick ( byte in ton ) is
                                    ;  procedură cu rol de sesizare al apăsării butoanelor
  for 20 loop
   pin_d1 = high
   delay_200us ( ton )
   pin_d1 = low
   delay_200us ( ton )
  end loop
end procedure
```

-- ************************
-- **PROGRAMUL PRINCIPAL**
-- ************************

```
semafor == verde     -- porneşte cu meniul iniţial
display_first        -- afişează meniul LCD iniţial
intcon_gie = high    -- setează  întreruperile globale
intcon_t0ie = high
                 -- setează întreruperile tmr0, foarte important, fără ele nu pot fi citite butoanele
forever loop
```

```
if semafor == verde  then        -- testează meniul iniţial
  if but1 then tick ( 2 ) but1 = low display_tmp
                                 -- afişare timp
  elsif but2 then tick ( 2 ) but2 = low display_pwr
                                 -- afişare putere
    semafor = 5                  -- se sare în meniul de programare putere
    hd44780_clear                -- iniţializarea LCD
    hd44780_line1                -- cursorul LCD este pus în linia1 poziţia 0
    hd44780 = "P" hd44780 = "w" hd44780 = "r"  hd44780 = ":"
    display_pwr_value                   -- afişarea valorii puterii
    display_grafic
          -- afişarea caracterelor grafice predefinite pentru UP şi DOWN (fig.3-10)
   elsif but3 then tick ( 2 ) but3 = low
    display_tratament                   -- afişează meniul de tratament
   elsif but4 then                      -- buton inactiv, nu face nimic în acest meniu
  end if
end if
```

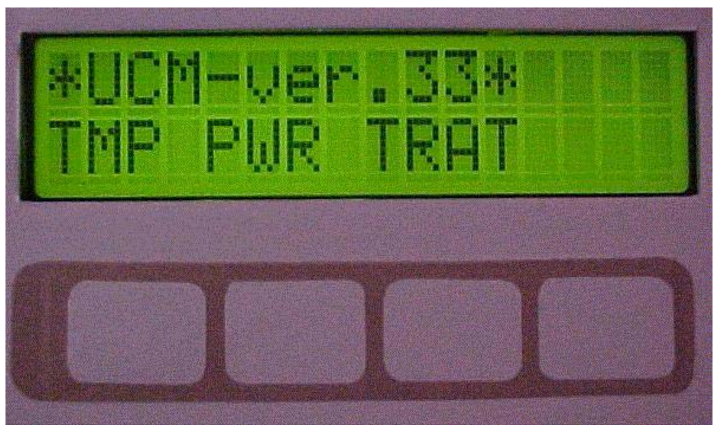


fig.3-8 Meniul display_first şi butoanele funcţionale but1, but2 şi but3, semafor = verde

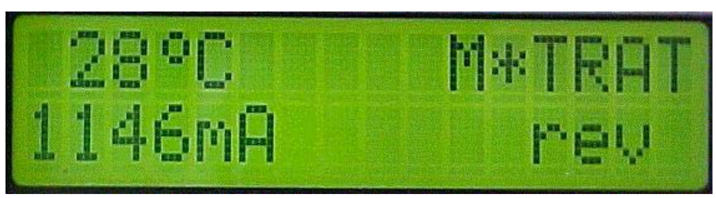


Fig.3-9 Meniul măsură, numai but4 este activ, semafor = roşu

```
-- -------------------------
-- MASURA
-- -------------------------
if semafor == rosu then                 -- modul de măsură
 if but4 then tick ( 2 ) but4 = low
                                        -- resetarea variabilei but4 pentru citirea următoare
  dispclk_flag = off                    -- ceasul este pornit dar nu este afişat
  a_flag = on                           -- porneşte afişarea curentului anodic
  dispclk_flag = on                     -- afişează ceasul
  measure_flag = on                     -- măsurătoarea analogică s-a terminat
```

```
   save_second = second
   save_minute = minute     -- salvează ceasul în variabile locale pentru o utilizare viitoare
   display_tratament        -- afişează ecranul LCD de tratament
   display_stby_off         -- cu funcţia stby şi off a butoanelor
  end if
end if
```



fig.3-10 Meniul programare timp, but1, but2 şi but4 sunt active, semafor = galben

```
-- ------------------------------
-- PROGRAMARE  TIMP
-- ------------------------------
if semafor == galben then   -- modul de programare al timpului
   if ! write then           -- citirea eeprom-ului se face doar prima dată şi …
    eeprom_read_877 ( min_address, minute )
   end if                    -- … se citeşte ultima valoare memorată în eeprom…
                             -- …la salvarea precedentă
   display_minute            -- afişează minutele
  if but1 then tick ( 2 ) but1 = low   -- UP, incrementare
       write = on            -- activează flagul de salvare în eeprom
       minute = minute + 1   -- incrementează minutele
       if minute > 59 then minute = 59 end if -- nu depăşi 59
       display_minute        -- afişează minutele
  elsif but2 then tick ( 2 )  but2 = low – DOWN, decrementare
        write = on           -- activează salvarea în eeprom
        minute = minute - 1  -- decrementează minutele
        if minute == 255 then minute = 0 end if -- 0 cea mai mică valoare
        display_minute       -- afişează minutele
  elsif but4 then tick ( 2 ) but4 = low   -- memorează minutele în eeprom
       eeprom_write_877 ( min_address, minute ) -- salvează şi…
       write = off           -- …dezactivează salvarea în eeprom
       display_first         -- afişează meniul iniţial corespunzător lui semafor = 0
       semafor == verde      -- salt în meniul iniţial
  end if
end if
…
-- alte rutine care sunt rulate fie prin setare de flaguri, fie condiţionate de semafor
end loop
```

Pentru înţelegerea corectă a secţiunilor de program corespunzătoare fiecărei culori a semaforului, am fotografiat ceea ce este afişat în mod real pe LCD-ul instalaţiei unde rulează acest firmware; este vorba de o instalaţie de tratament de mare putere în câmp de microunde. Am preferat să nu detaliez ceea ce se ascunde în spatele unor rutine în favoarea înţelegerii algoritmului cu semafor de meniu. Semaforul este definit ca o constantă. După ce este testat (*if* semafor == x *then*…) valoarea lui se poate schimba asigurând saltul în altă

ramură a programului. Practic se pot utiliza semafoare cu până la 255 de valori, însă în mod real nu se depăşesc valori mai mari de 10..20. Dacă sunt suficiente două stări logice ale semaforului, acesta se transformă într-un flag iar variabila ce-l defineşte devine de tip bit. Un exemplu (nu foarte simplu) de implementare a semaforului cu meniuri se găseşte pe CD:\jal_examples.

3.3.3 Matrici de butoane sau "keypad"

Cu toate că butoanele funcționale sunt o opțiune interesantă, există situații când este improprie utilizarea lor. Nu v-a exasperat nici o dată navigarea prin meniurile telefonului dvs. celular ? Cea mai cunoscută modalitate de interfațare (de pe vremea când bunica era fată şi calculatoarele aveau microprocesor 8080) este matricea de 4 linii şi 4 coloane (fig.3-11) pe care se interfațează o tastatură de tip keypad (matrice de butoane). Pentru citirea tastaturii (fig.3-11), coloanele vor fi setate ca ieşiri, cuvântul scris conținând bitul 0, baleiat secvențial pe fiecare din RB4, RB5, RB6 şi RB7. Rezistențele de 220 de ohmi sunt necesare numai dacă coloanele respective sunt utilizate secvențial şi pentru comanda unor afişoare. Rândurile RB0, RB1, RB2 şi RB3 sunt setate ca intrări, portul B având rezistențele de pull-up activate din registrul *option.* După fiecare baleiere a coloanelor are loc o citire a rândurilor, astfel că în situația unei taste apăsate, rândul corespunzător este în 0 logic numai pentru o anumită configurație a coloanei. Citirea tastelor se face cu acelaşi mecanism de anulare a zgomotelor de comutație descris anterior. Singurul impediment al acestui mod de citire este situația când sunt apăsate simultan două taste. Din program, utilizatorul poate opta pentru validarea ultimei taste apăsate, sau poate iniția un mecanism de schimbare a funcției rândurilor şi coloanelor între ele pentru determinarea exactă a tastei apăsate corect în detrimentul celei apăsate din greşeală (se consideră că tasta greşită stă apăsată un timp mai scurt decât cea corectă). In acestă situație, rezistențele de 10K din intrări trebuie anulate. De exemplu fie A (RB3-RB7) tasta apăsată. Pentru combinația de comandă 0b_0111_xxxx va rezulta codul tastei 0b_xxxx_0111, unde valoarea bitului x este nesemnificativă pentru exemplul de față. Dacă se apasă din greşeală şi tasta 0 (RB2-RB7), citirea acesteia se va face numai pentru comanda: 0b_1011_xxxx iar rezultatul va fi 0b_xxxx_1011. Deşi metoda pare mare consumatoare de resurse hardware, este frecvent utilizată deoarece permite multiplexarea funcțiilor unei jumătății din portul B, fie rânduri fie coloane, pentru un alt periferic cum ar fi afişajul cu LCD sau afişajul cu LED-uri cu şapte segmente. Programul Jal care citeşte o tastatură şi returnează valoarea ASCII a ultimei taste apăsate vă încântă privirea imediat:

```
var volatile bit R1 is pin_b0      ; R -- rînduri
var volatile bit R2 is pin_b1
var volatile bit R3 is pin_b2
var volatile bit R4 is pin_b3
var volatile bit C1 is pin_b4      ; C -- coloane
var volatile bit C2 is pin_b5
var volatile bit C3 is pin_b6
var volatile bit C4 is pin_b7

port_b_low_direction = all_input
```

```
port_b_high_direction = all_output
var byte key

procedure keyboard is
port_b_high = 0b_1110
if ! C1 then key = "1" end if ; "1" este simbolul ASCII 1 şi nu cifra 1 !
if ! C2 then key = "2" end if
if ! C3 then key = "3" end if
if ! C4 then key = "C" end if
```

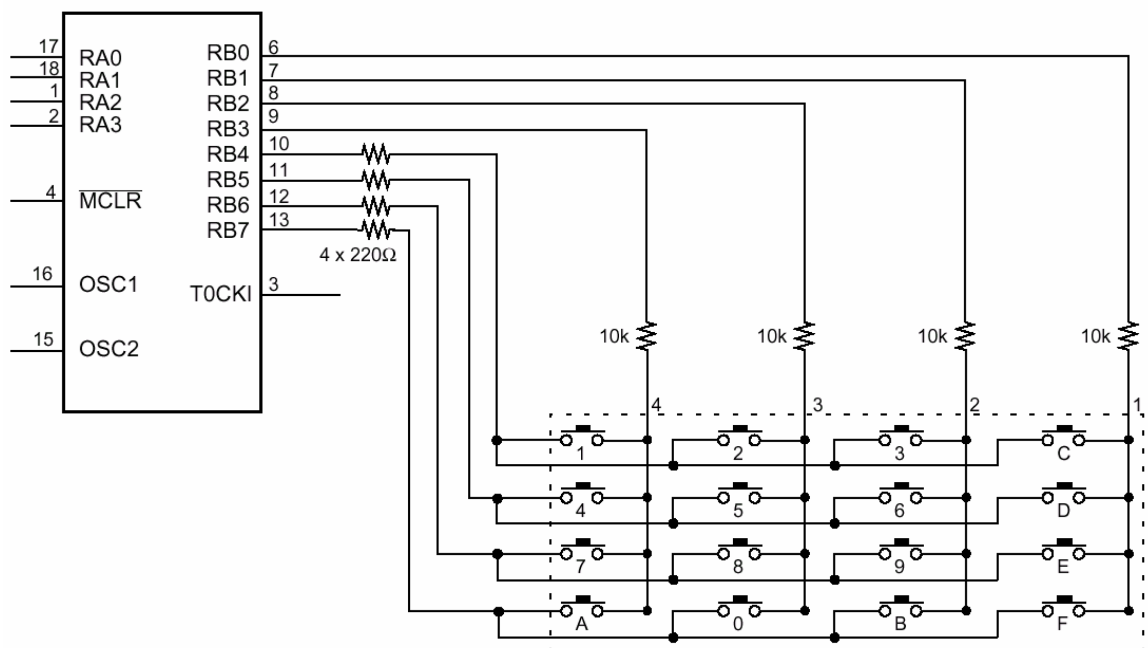


Fig.3-11 Matricea de butoane

```
port_b_high = 0b_1101
if ! C1 then key = "4" end if
if ! C2 then key = "5" end if
if ! C3 then key = "6" end if
if ! C4 then key = "D" end if
port_b_high = 0b_1011
if ! C1 then key = "7" end if
if ! C2 then key = "8" end if
if ! C3 then key = "9" end if
if ! C4 then key = "E" end if
port_b_high = 0b_0111
if ! C1 then key = "A" end if
if ! C2 then key = "0" end if
if ! C3 then key = "B" end if
if ! C4 then key = "F" end if
end procedure
```

3.3.4 Metoda de interfaţare derivativă

Această metodă se bazează pe proprietatea unui condensator polarizat cu o tensiune continuă, de a permite parcurgerea sa de un curent tranzitoriu de încărcare din momentul polarizării iniţiale şi până când încărcarea a luat sfârşit. Dacă pe un singur pin al microcontrolerului se conectează un buton în modul clasic [fig.3-12a] şi un număr de **n** butoane prin **n** capacităţi de valoare diferită, durata de timp rezultată în urma apăsării fiecărui buton va fi diferită şi prin determinarea precisă a acesteia, se poate identifica butonul apăsat. Se observă că butonul S1 este conectat direct pe pinul a0 şi apăsarea lui va produce un nivel logic 1 atât timp cât este menţinut apăsat. Apăsarea butonului S2 va încărca condensatorul C1. Iniţial pinul a0 va avea un nivel logic 1 urmat de scăderea cu o alură exponenţială a potenţialului intrării a0, încărcarea totală a condensatorului putând fi considerată încheiată după o perioadă t = 3C1R1, adică după aproximativ 140mS. Perioada critică de citire a butonului este ceva mai mică decât constanta de timp a circuitului, adică aproximativ 40…50mS datorită trigerului schmitd pe care fiecare intrare îl are. După ce tasta a revenit în poziţia iniţială, condensatorul se descarcă pe rezistenţele R1, R2 cu o constantă de timp aproximativ dublă decât la încărcare, motiv pentru care butonul S2 va fi activ numai în apăsări succesive având duratele între ele de cel puţin 280…300mS.

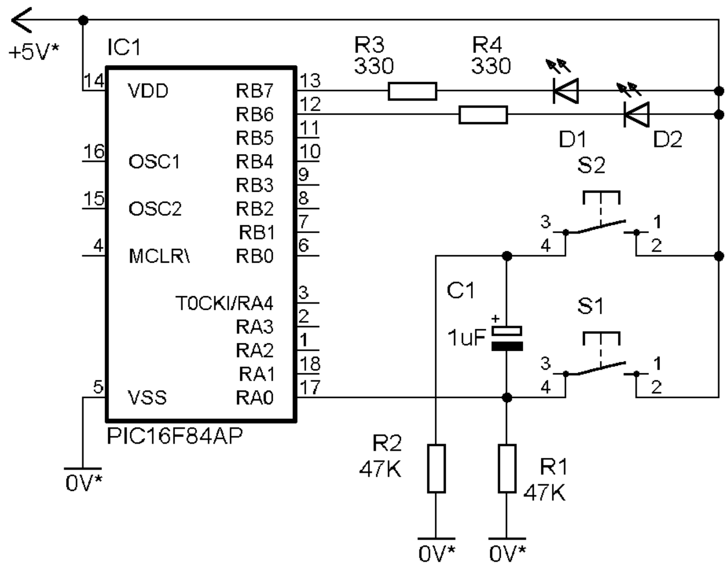


Fig.3-12a Citirea derivativă a unui buton

Există şi metoda inversă, în care încărcarea unui singur condensator se face prin rezistenţe diferite, comutate în circuit de n butoane [fig.3-12b]. Pinul I/O descarcă iniţial condensatorul C prin rezistenţa de 220 ohmi. Apoi pinul este setat ca intrare de comparator şi măsoară timpul de încărcare al condensatorului C până la atingerea tensiunii de referinţă existente pe cealaltă intrare de comparator. Apăsând butonul n, constanta de timp de încărcare va fi nR*C şi aceasta trebuie măsurată.

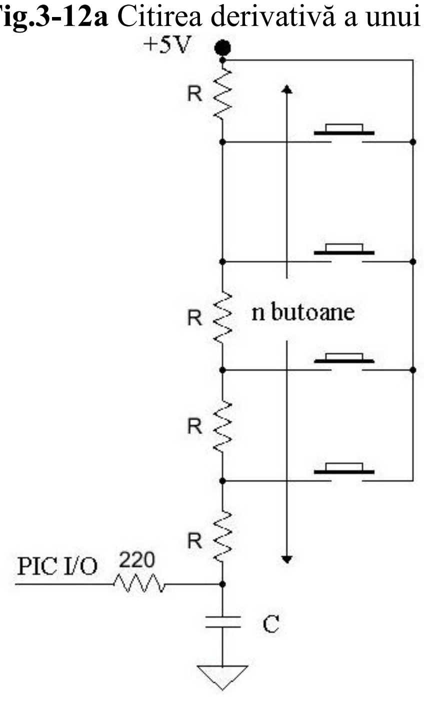
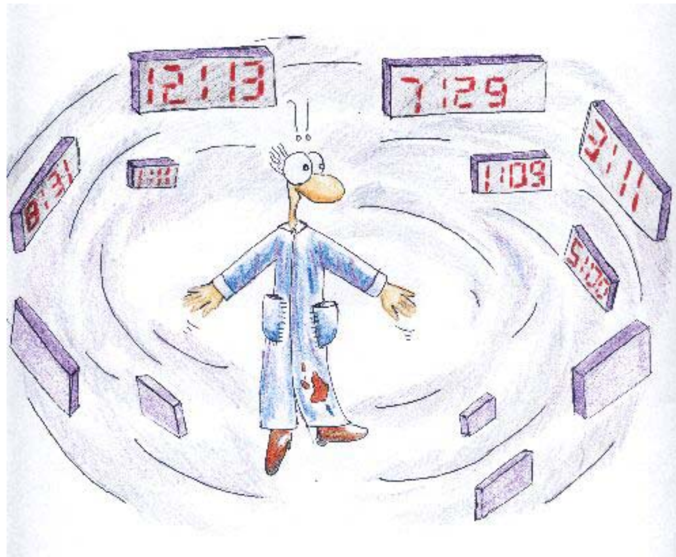


Fig. 3-12b Conectarea a n butoane pe acelaşi pin

3.4 Interfaţarea afişajelor cu 7 segmente



Oricine a văzut cel puţin o dată în viaţă un astfel de afişaj. Din punct de vedere al conexiunii LED-urilor ce formează segmentele dispozitivului, se întâlnesc afişoare cu anod şi cu catod comun, denumirea provenind de la tipul terminalului diodelor care se conectează împreună şi care apare în capsula afişajului ca terminal de alimentare. Din punctul de vedere al posibilităţilor de afişare există dispozitive numerice (afişează cifrele 0…9) şi alfanumerice (afişează inclusiv litere). După culoarea segmentelor, cele mai întâlnite afişaje au culoarea roşie sau verde însă există mai recent afişaje cu nuanţă albă sau albastră. După modul în care se conectează mai multe afişaje la dispozitivul de comandă sau microcontroler există varianta cu multiplexare sau cu polarizare independentă. Este dispozitivul cel mai ieftin existent la ora actuală pe piaţă, utilizat pentru a transmite un mesaj (preţ, greutate, timp, temperatură sau informaţie de uz general).

3.4.1 Afişaj cu 7 segmente cu polarizare independentă

Biblioteca jseven.jal este destinată afişării cifrelor 0…9 pe afişaje numerice cu anod sau catod comun, cu interfaţare directă la microcontroler (fără a utiliza circuite decodoare gen 74SN47), cât şi a literelor care pot fi formate pe un afişaj numeric (A, b, c, C, d, E, F, r, H, i, L, o, P, u, U). Schema tipică de interfaţare este simplă şi include doar rezistenţele de limitarea a curentului de segment, dimensionate în acelaşi mod ca şi pentru comanda unui LED standard. Afişajele gigant cu LED-uri folosesc două sau trei LED-uri înseriate pentru fiecare segment. Este foarte posibil ca aceste afişaje să necesite tensiuni mai mari de 5V pentru a aprinde segmentele cu o luminozitate acceptabilă, motiv pentru care datele de catalog referitoare la curentul maxim pe segment şi căderea de tensiune pe segment trebuiesc cunoscute (sau cel puţin descoperite prin măsurători). Curentul maxim pe segment nu depăşeşte 30…35mA nici pentru afişajele gigant (10 cm înălţime). Depăşirea curentului maxim un timp scurt duce la viraj de culoare şi încălzirea pronunţată a afişajului (de exemplu un afişaj cu nuanţa verde va vira în galben la un curent cu 30% mai mare decât curentul maxim, apoi în oranj, după care se va arde!). Afişajele economice lucrează cu curenţi pe segment cuprinşi între 1…5mA însă au dimensiuni mici (sub 7.5mm înălţime). O caracteristică importantă a acestor afişaje este modul în care se reflectă iluminarea unui segment aprins în segmentele stinse adiacente şi este din păcate o variabilă ce depinde de producător şi nu poate fi determinată decât în procesul de testare-producţie având doar două valori: afişaj bun şi respectiv afişaj prost.

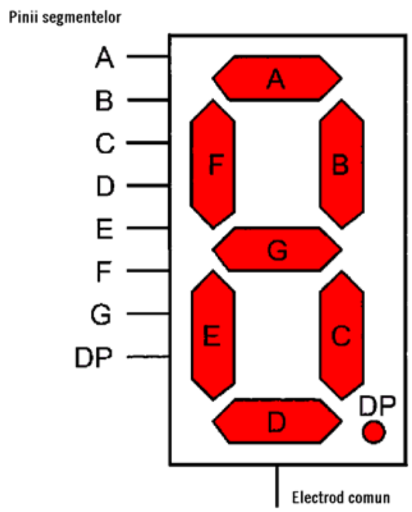


Fig.3-13 Numerotarea standardizată a segmentelor

In fig.3-13 este prezentată standardizarea numerotării segmentelor unui astfel de afişor. Presupunând ca dispunem de un afişaj cu catod comun, acesta se va lega la masă iar segmentele A…G prin intermediul a şapte rezistențe de 330 ohmi la ieşirile portului B ale microcontrolerului. Curentul maxim generat pe fiecare pin al PIC-ului spre masă nu trebuie să depăşească 25mA. Alegerea unui curent de lucru de 10…15mA este o opțiune ideală. Pentru un afişaj cu anod comun, electrodul comun se va conecta la +5V iar segmentele prin aceleaşi rezistențe de limitare la portul B al pic-ului. Curentul maxim acceptat de fiecare intrare a microcontrolerului este de maxim 20mA.

Un programel care va testa afişajul afişând cifre de la 0 la 9 şi litere de la A la F poate fi scris cu uşurință:

```
include 16f84_4
include jpic
include jseven
include jdelay
const byte catod_comun = 0
const byte anod_comun = 1
var byte bin = 0
var volatile byte tip_afişor
port_b_direction = all_output

 for 16 loop
  if tip_afişor == anod_comun then
   port_b = ! seven_from_digit ( bin )
  elsif tip_afişor == catod_comun then
   port_b =  seven_from_digit ( bin )
  end if
   delay_100mS( 5 )
   bin = bin + 1
 end loop
```

Simplitatea utilizării bibliotecii este evidentă. **Avantajul** interfațării este indubitabil: se poate regla intensitatea luminoasă prin modificarea valorii rezistențelor sau a tensiunii electrodului comun, opțiune importantă pentru afişaje ce funcționează în plin soare. Pentru afişoare economice de curent mic se poate folosi acceptabil o singură rezistență în anodul sau catodul dispozitivului cu o uşoară schimbare a luminozității în funcție de cifra afişată. Utilizarea unei singure diode zener în locul rezistenței comune de balast este o opțiune interesantă, având ca rezultat intrarea în limitare a tranzistorului MOS intern circuitului de comandă prin micşorarea U_{DS}; luminozitatea afişajului rămânând neschimbată indiferent de numărul de segmente aprinse. **Dezavantajul** conectării unui număr mai mare de afişoare în acest mod este de asemenea evident, necesită n x 7 pini disponibili în microcontroler, unde

n este numărul de afişoare. Pentru n = 6 (cazul unui frecvențmetru numeric) avem nevoie de 42 de pini, mai mulți decât dispune cel mai puternic microcontroler flash midrange ! Soluția acestei probleme este şi ea destul de simplă:

- Utilizarea unui decodor bcd/7segmente (nu mai poate afişa literele A…F) reduce numărul de pini necesari pentru fiecare afişaj la 4.
- Utilizarea multiplexării alimentării electrozilor comuni ai afişajelor şi conectarea în comun a segmentelor; pentru exemplu de mai sus, 6 afişoare vor necesita 6 + 7 = 13 pini de comandă în microcontroler.
- Utilizarea circuitelor specializate de afişare cu încărcare serială, de exemplu MMC22925/MMC22926, care rezolvă conectarea a 4 afişoare (sau mai multe prin cascadarea cipurilor), consumând doar 3 pini ai microcontrolerului.
- Utilizarea serializării cu regiştrii de deplasare şi conectarea afişajelor pe ieşirile acestora.

3.4.2 Multiplexarea, ceas de precizie cu afişaje cu 7 segmente

"Teoria ca teoria, da' practica ne omoară !" O teorie bună, aplicată corect în practică, va da rezultate imediat. Exemplul următor infirmă zicala populară de mai sus. Este vorba de un ceas cu format 24 de ore, a cărui setări utilizează trei butoane miniatură (push-buttoane), un difuzor de tip buzzer (fără oscilator incorporat) cu impedanța de cca. 47 ohmi care are funcție dublă de semnalizare a rolului butoanelor, de "cuc" la ore predefinite de utilizator sau de alarmă pentru deşteptare. Adică cititorul poate seta din program când să cânte cucul, numai la orele din zi sau din noapte pe care le doreşte şi nu aşa cum face un ceas obişnuit, destul de plicticos, la toate orele exacte! Afişarea foloseşte principiul multiplexării, cunoscut cititorului pasionat de petrecerea timpului liber în sălile de cinematograf: dacă frecvența de derulare a unor imagini statice prin fața ochilor subiectului este mai mare de 24 de cadre pe minut, rezultatul va fi impresia unei imagini cu mişcare continuă. In cazul nostru imaginea este realizată prin aprinderea succesivă a unui singur afişor din cele patru existente, cu o asemenea viteză încât impresia observatorului va fi aceea că toate cele 4 afişaje sunt aprinse simultan. Generarea bazei de timp de o secundă (Real Time Clock) se poate face prin mai multe metode, fie utilizând un circuit integrat extern care generează timingul de o secundă, ceas şi calendar, fie utilizând baza de timp proprie (TMR0) a microcontrolerului şi făcând corecții la intervale mai mari de timp (minute), fie utilizând o derivată a metodei Bressenham prezentată şi în CD:\ tutor\one_sec.htm

Ceasul foloseşte afişaje cu anod comun şi o schemă derivată din nota de aplicație AN615, rezistențele R4 au valoarea de 3k9 iar rezistențele R5 de 5k6. Digitul U2 reprezintă minutele, U3 zecile de minute, U4 orele iar U5 zecile de oră. Rb0…Rb6 comandă segmentele de la A la F, asigurându-se o limitarea a curentului prin rezistențele de 47 ohmi. SW3 reglează minutele, SW1 reglează orele iar SW2 comută funcția lui SW1 şi SW3 din reglajul ceasului în reglajul alarmei deşteptătoare.

```
include 16f84_4                -- biblioteci incluse în program:
include jpic
include jseven
include jdelay
```

```
var byte sel_min_units = 0b_1110  -- variabilele utilizate în program
var byte sel_min_tens = 0b_1101
var byte sel_hours_units = 0b_1011
var byte sel_hours_tens = 0b_0111
var byte sel_blank = 0b_1111
port_a_direction = all_output
pin_b7_direction = output
var byte sec = 0
var byte min_units = 0
var byte min_tens = 0
var byte hours_units = 2
var byte hours_tens = 1
var byte amin_units
var byte amin_tens
var byte ahours_units
var byte ahours_tens
var bit but_flag = low
var bit alarm_flag = low
var bit cucu_flag = low
var byte roman_hi = 0x_0f
var byte roman_mid = 0x_43
var byte roman_lo = 0x_40

clear_watchdog
option = 0x_88
tmr0 = 0
asm bsf intcon_gie               -- intcon_gie = on, intreruperi globale activate
asm bsf intcon_t0ie              -- intcon_t0ie = on, intreruperi ale tmr0 activate

procedure time is
  asm incf sec, f
if sec == 60 then asm incf min_units, f  asm clrf sec
end if                                  -- testare 60 secunde, incrementare minute
if min_units == 10 then asm clrf min_units
 asm incf   min_tens, f
end if
if min_tens == 6 then asm clrf min_tens
 asm incf hours_units, f
end if                                  -- testare 60 minute, incrementare ore
if hours_units == 10 then asm clrf hours_units
 asm incf hours_tens, f
end if
if hours_tens == 2 then
  if hours_units == 4 then asm clrf hours_tens
   asm clrf hours_units
  end if                      -- testare 24 ore
end if
end procedure
procedure isr is              -- procedură de tratare a întreruperilor
pragma interrupt              -- salt la adresa 04h
 assembler
  local out
     tstf roman_mid           -- roman_mid = 0 ?
     skpnz
```

```
      decf roman_hi, f         -- roman_mid = 0 deci decrementează roman_hi
      decfsz roman_mid, f      -- roman_mid = roman_mid - 256
      goto out                 -- dacă nz (not zero)nu a trecut încă o secundă
      tstf roman_hi            -- test roman_hi
      skpz                     -- dacă z (zero ) atunci roman_hi şi roman_mid sunt 0
      goto out                 -- dacă nz atunci nu a trecut încă o secundă
      movlw 0x_0f
      movwf roman_hi           -- încarcă msb
      movlw 0x_42
      movwf roman_mid          -- încarcă mid
      movlw 0x_40
      addwf roman_lo, f        -- *1) adună cu restul aflat deja în lsb
      skpnc                    -- skip if not carry, salt peste instrucțiunea următoare
      incf roman_mid, f        -- dacă este carry roman_lo a depăşit 255, incrementează…
                               -- …roman_mid
      call time                -- cheamă taskul utilizator (procedura time)
out: bcf intcon_t0if           -- resetează flagul de întreruperi
  end assembler
end procedure
```

Sunt necesare câteva lămuriri privind algoritmul de generare al ceasului de timp real utilizând metoda Bressenham. După cum am putut observa în capitolul 3.3 nu se pot obține orice valori pentru intervalele de timp generate din TMR0. Din această cauză se utilizează un artificiu care transcrie valoarea lui TMR0 într-un număr compus din trei octeți. De exemplu, presupunând că dispunem de un cristal de cuarț având valoarea de 7.37280 MHz, valoarea reală a frecvenței procesor va fi 7.37280/4 = 1.8432 MHz. Traducând această valoare în format hexazecimal vom avea 1843200 = 1C2000h; ceea ce înseamnă că e nevoie de 1843200 tacți pentru trecerea unei secunde. Valoarea regiştrilor din rutina de întreruperi este: hi = 0x1C, mid = 0x20, lo = 0 iar valoarea definită la începutul programului cu 256 (100h) mai mare: hi = 0x1C, mid = 0x21, lo = 0. Fiecare întrerupere generată va scădea 256 din variabila de 24 de biți (3 octeți) utilizând un algoritm rapid de calcul. Chiar dacă rezultatul scăderii este un număr negativ, valoarea acestuia rămâne memorată în regiştrii, asigurând adunării inițiale *1) o eroare extrem de mică pe termen lung.

```
procedure tick ( byte in ton ) is          -- procedură de avertizare sonoră
  for 20 loop
   pin_b7 = high
   delay_200us ( ton )
   pin_b7 = low
   delay_200us ( ton )
  end loop
end procedure

procedure button_minutes is                -- citirea butonului de reglaj minute
port_a = sel_blank                         -- şterge afişajul, se citesc butoane !
pin_b4_direction = input
 if ( ! pin_b4 ) then                      -- dacă e low atunci
  tick ( 2 ) asm incf min_units, f         -- min_units = min_units + 1
   if min_units == 10 then asm clrf min_units
    asm incf min_tens, f
   end if
```

```
   if min_tens == 6 then asm clrf min_tens
    asm incf hours_units, f
   end if                            -- testează depăşirea numărătorului de minute
 delay_10mS ( 5 )                    -- delay pentru citirea succesivă a butonului
 end if
pin_b4_direction = output            -- s-a terminat citirea butonului, urmează afişarea
end procedure

procedure button_hours is            -- citirea butonului de reglaj ore
port_a = sel_blank                   -- stinge afişajul
pin_b6_direction = input             -- intrare
 if ( ! pin_b6 )   then              -- citeşte butonul
  tick ( 2 )                         -- fă un tick să auzim şi noi că am apăsat
  asm incf hours_units, f
   if hours_units == 10 then asm clrf hours_units
    asm incf hours_tens,f
   end if
   if hours_tens == 2 then
    if hours_units == 4 then asm clrf hours_tens
     asm clrf hours_units
    end if                           -- testează depăşirea numărătorului de oră
   end if
  delay_10mS ( 5 )                   -- întârziere necesară pentru citirea butonului
 end if
pin_b6_direction = output            -- dezactivează butonul şi activează afişarea
end procedure

procedure button_alarm is
port_a = sel_blank                   -- stinge afişajul
pin_b5_direction = input             -- selectează funcția butonului
 if ( ! pin_b5 )  then               -- citeşte butonul
   if ! but_flag then                -- dacă se execută un preset al alarmei
    tick ( 1 )
     eeprom_put(1, min_units)        -- memorează valoarea actuală a ceasului
     eeprom_put( 2, min_tens )
     eeprom_put( 3, hours_units )
     eeprom_put( 4, hours_tens )
     but_flag = ! but_flag
   elsif but_flag then               -- dacă se execută un preset al ceasului
    tick ( 3 )
     eeprom_put( 5, min_units )      -- memorează alarma
     eeprom_put( 6, min_tens )
     eeprom_put( 7, hours_units )
     eeprom_put( 8, hours_tens )
     eeprom_get( 1, min_units )      -- încarcă valoarea anterioară a ceasului
     eeprom_get( 2, min_tens )
 -- acest reglaj trebuie făcut în mai puțin de 1 minut pentru a nu pierde valoarea ceasului
     eeprom_get(3, hours_units)
     eeprom_get( 4, hours_tens )
    but_flag = ! but_flag            -- resetează flagul pentru următoarea operație
   end if
  delay_10mS ( 5 )                   -- întârziere necesitată de butoane
```

```
  pin_b5_direction = output          -- dezactivează butonul şi activează afişarea
 end if
end procedure

procedure display is
  port_b_direction = all_output   -- pregăteşte pentru afişare
  port_a = sel_min_units          -- multiplexează min_unit şi
  port_b = ! seven_from_digit( min_units )   -- afişează
  delay_1ms( 3 )                  -- menţine afişarea 3mS pentru o bună vizibilitate
  port_a = sel_min_tens           -- multiplexează min_tens şi
  port_b = ! seven_from_digit( min_tens ) -- afişează
  delay_1ms( 3 )                  -- menţine afişarea pentru o bună vizibilitate
  port_a = sel_hours_units        -- etc.
  port_b = ! seven_from_digit( hours_units )
  delay_1ms( 3 )
 if hours_tens == 0 then port_a = sel_blank
                                  -- dacă cel mai semnificativ digit
 elsif hours_tens > 0 then        -- e zero stinge afişajul, dacă nu afişează
   port_a = sel_hours_tens
   port_b = ! seven_from_digit( hours_tens)
   delay_1ms( 3 )
 end if
end procedure

procedure cucu ( byte in cuchours_tens,
                 byte in cuchours_units,
                 byte in cucmin_tens, byte in cucmin_units )   is
 if ( cucmin_units == min_units ) &       -- sincronizare la ora exactă ?
    ( cucmin_tens == min_tens ) &
    ( cuchours_units == hours_units ) &
    ( cuchours_tens == hours_tens ) then
   if ! cucu_flag then                     -- da, fă gălăgie !
    for 5 loop tick ( 1 ) tick ( 2 ) end loop
    cucu_flag = ! cucu_flag                -- gata, opreşte-te !
   end if
 end if
  if min_units > cucmin_units then
                           -- a trecut un minut, resetează flagul pentru următorul
   cucu_flag = low
  end if
end procedure

procedure alarm is
 eeprom_get( 5, amin_units )
 eeprom_get( 6, amin_tens )               ; citeşte valoarea prog.
 eeprom_get( 7, ahours_units )
 eeprom_get( 8, ahours_tens )
 if ( amin_units == min_units ) &         ; şi compară cu valoarea actuală
    ( amin_tens == min_tens ) &
    ( ahours_units == hours_units ) &
    ( ahours_tens == hours_tens) then
    port_a = sel_blank            ; şterge afişajul
   if ! alarm_flag then           ; fă gălăgie şi afişează că altfel nu vedem nimic !
```

```
    for 50 loop tick ( 1 ) tick ( 2 ) display tick ( 3 )
    end loop
    alarm_flag = ! alarm_flag     ; opreşte gălăgia
   end if
  end if
  if min_units > amin_units then  ; şi curăţă flagul după un minut
   alarm_flag = low
  end if
end procedure

 forever loop                     ; programul principal
  cucu ( 0, 8, 0, 0 )             ; cântă cucul la ora şi minutul programat
  button_minutes                  ; citeşte butonul de reglaj minute
  cucu ( 0, 9, 0, 0 )             ; etc
  button_hours
  cucu ( 1, 0, 0, 0 )
  button_alarm
  cucu ( 1, 1, 0, 0 )
  cucu ( 1, 2, 0, 0 )
  cucu ( 1, 3, 0, 0 )
  alarm                           ; sună soneria dacă e cazul
  cucu ( 1, 4, 0, 0 )
  display                         ; şi afişează ceva că doar e un ceas !
  cucu ( 1, 5, 0, 0 )
  cucu ( 1, 6, 0, 0 )
 end loop
```

Un element eseţial corectei funcţionări a ceasului este acumulatorul Ni-Cd cu tensiunea nominală de 3.6V care trebuie menţinut în tampon cu alimentarea de la reţea, după stabilizatorul local de +5V. Aceste elemente nu sunt figurate integral în schema electronică, modul posibil de conexiune fiind prezentat în capitolul 4.13. Rezultatul schemei electronice şi al programului de mai sus poate fi văzut în imaginea din fig.3-15.



Fig.3-15 Ceas electronic montat pe placă prototip

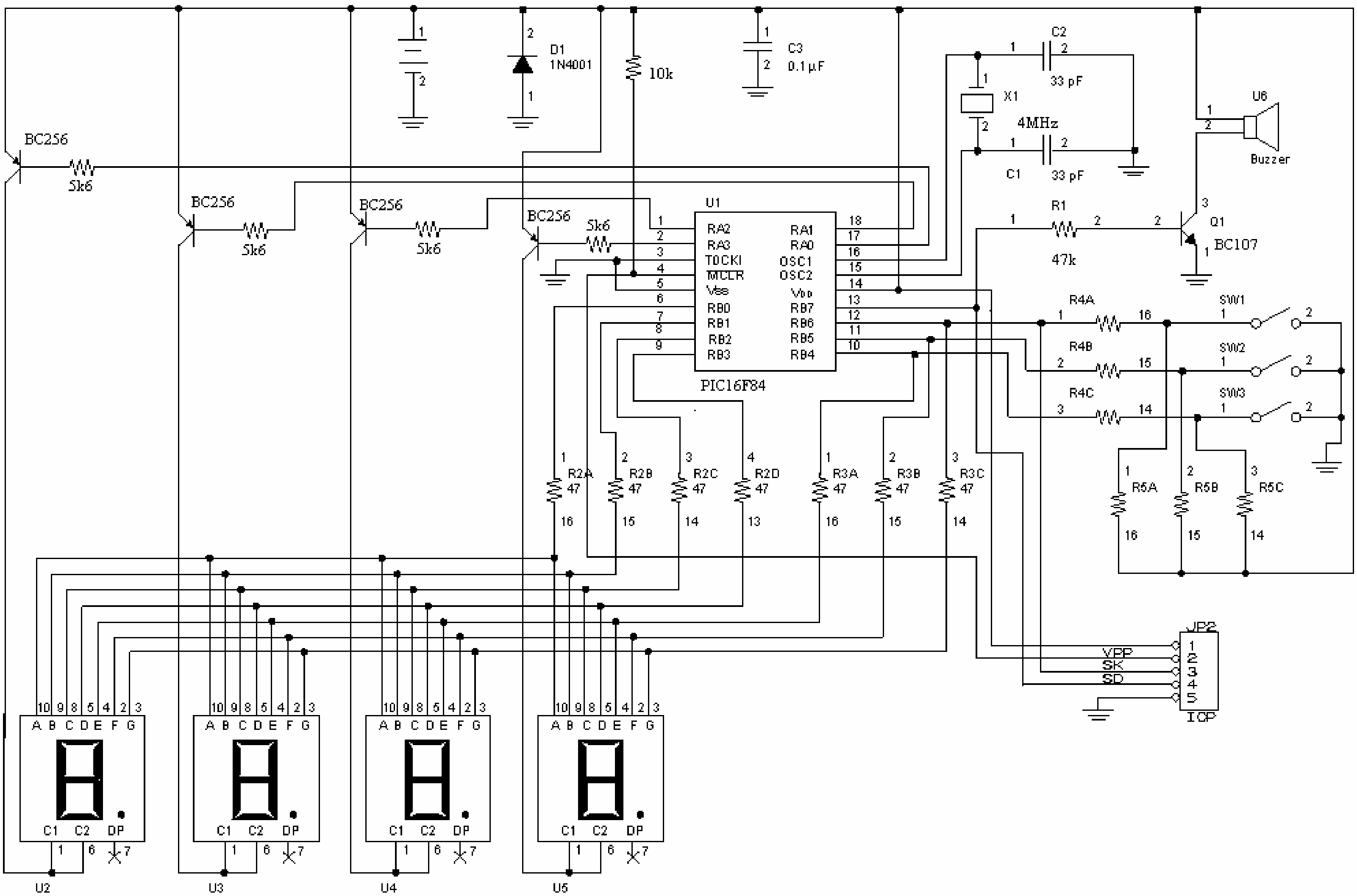


Fig.3-14 Ceas electronic – schema de principiu

Cele 4 afisoare U2, U3, U4 si U5 au anodul comun fiind comandate multiplexat prin tranzistoare PNP de orice tip. D1 protejeaza circuitul la montarea inversa a acumulatorului Ni-Cd. Tensiunea nominala a acumulatorului e bine sa fie de 4.8V (4 elemente de 1.2V in serie), desi la 3.6V ceasul este inca functional. Circuitul de oscilator X1, C1, C2 se monteaza in imediata apropriere a microcontrolerului utilizand conexiuni scurte. JP2 este conectorul de programare in circuit (ICSP). Prin acest conector se realizeaza programarea microcontrolerului. Butoanele SW1, SW2, SW3 realizeaza programarea ceasului si a soneriei la ora dorita.

3.4.3 Dispozitive de afişare independente CMOS

Dezavantajul major al multiplexării din microcontroler este, după cum aţi observat, consumul mare de resurse hardware, respectiv de porturi având rol de ieşiri. O primă analiză ne conduce la soluţia utilizării decodoarelor BCD/7 segmente (CDB446, CDB447, MMC4543 sau MMC4511) care reduce numărul pinilor utilizaţi pentru comanda segmentelor de la 7 la 4, numărul pinilor de comandă ai electrozilor comuni (anod sau catod) rămânând neschimbat. Există şi circuite specializate care conţin inclusiv comanda acestora respectiv MMC22925 şi MMC22926. O aplicaţie care evidenţiază avantajele (şi dezavantajele) acestei metode este descrisă în fig.3-16. Se observă numărul mic de pini (doar 5) consumaţi din microcontroler pentru a interfaţa 5 butoane (din care 4 utilizează algoritmul prezentat în cap.3.3.1) şi numărătorul de 4 digiţi cu decodare BCD/7segmente şi multiplexare MMC22925/MMC22926 care comandă afişajul de 3 digiţi cu catod comun. Deoarece au fost necesari doar 3 digiţi, s-au utilizat doar cei mai semnificativi (Qs2…Qs4), digitul cel mai nesemnificativ rămânând neconectat. După realizarea interfaţării au rămas în total, 6 pini disponibili din microcontroler (Ra0…Ra4 şi Rb5). Pinii Rb0 şi Rb1 sunt utilizaţi doar ca ieşiri putând fi intrări în faza în care nu se afişează. Descrierea detaliată a circuitului CMOS se găseşte în [3]. Semnalul de tact se aplică pe intrarea CK, tactul fiind activ pe front căzător. Funcţionarea circuitului este evidenţiată în tabelul următor:

	Re**S**e**T**	**L**atch **E**nable	**D**isplay **S**elect
0	şterge	memorează	afişează latch-urile
1	numără	transparent	afişează numărătoarele

Circuitul este alcătuit din 4 numărătoare divizoare cu 10, urmate de *latch*-uri şi decodoare pentru 7 segmente cu catod comun. RST este asincron, activ pe 0. Se observă că LE şi DS sunt conectate împreună. Acest lucru înseamnă că un nivel logic *high* pe acestea va permite modificarea valorii afişate pe display, în timp ce un nivel logic *low* va memora afişarea ultimei valori păstrate în *latch*. Această conectare economică este necesară pentru a corecta inexistenţa funcţiei de numărare inversă (*count down*) a acestui circuit, printr-o şmecherie: numărarea directă (*count up*) are loc generând tact pe intrarea CK, fiecare tact va incrementa numărătoarele, în timp ce numărarea inversă implică un RST urmat de o numărare directă până la valoarea iniţială din care scădem decrementarea dorită. Presupunând că afişarea actuală este 860 şi dorim o decrementare cu 1, se va genera un reset şi o incrementare cu 859. Această metodă software suplineşte o deficienţă a circuitului dar produce întârzieri mai mari la afişarea valorii decrementate, motiv pentru care utilizatorul trebuie să genereze o întârziere la incrementare dacă aplicaţia solicită timpi egali de incrementare/decrementare. Se observă de asemenea că apăsarea butonului B5 (a cărui funcţie este lăsată la discreţia utilizatorului) nu modifică ultima valoare afişată ci doar o memorează. Menţinerea apăsată o perioadă lungă de timp a butonului B5 (timp în care se operează modificări asupra numărătoarelor din MMC22925) urmată de relaxarea sa,

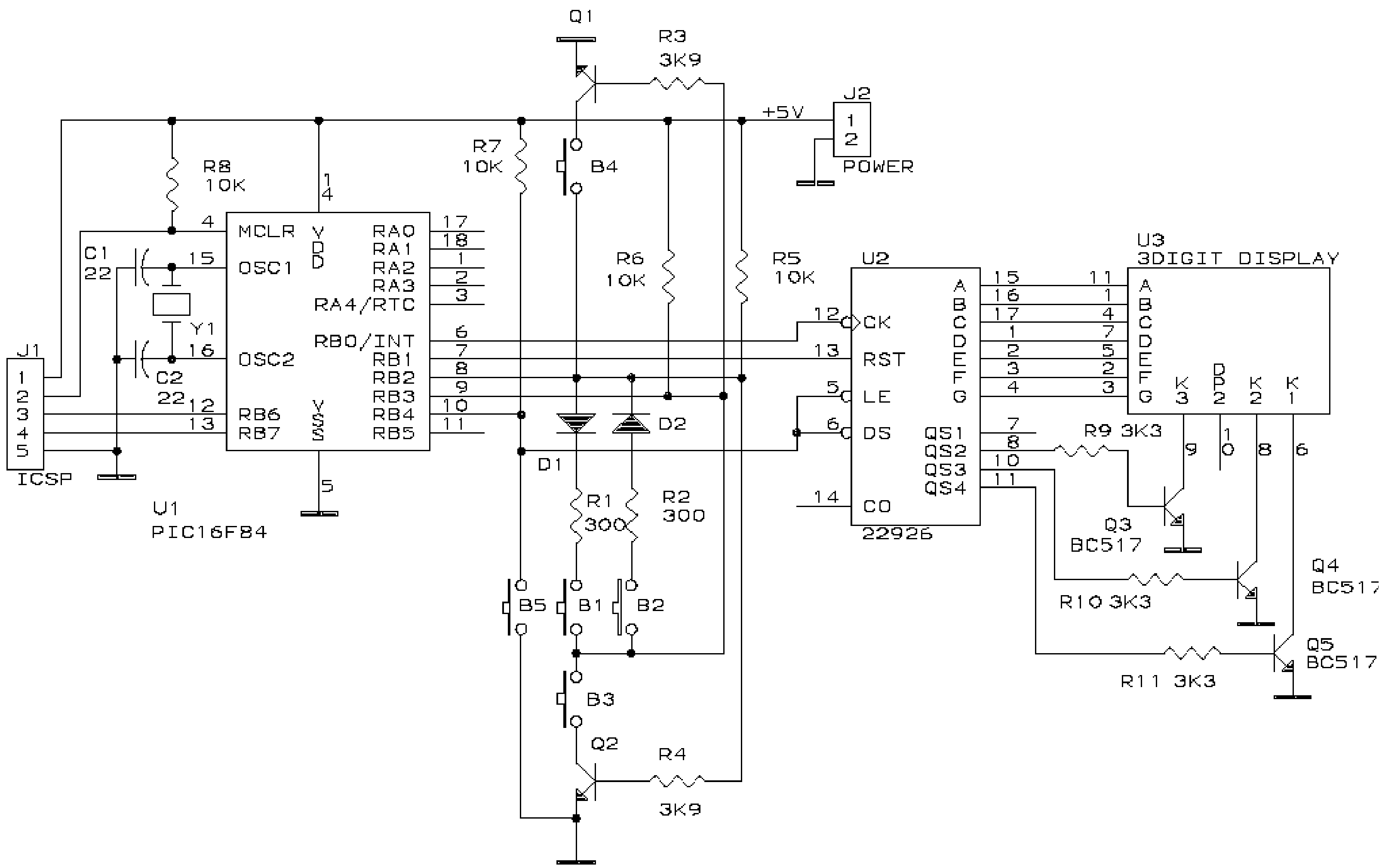


Fig.3-16 Interfaţarea unui dispozitiv de afişare cu multiplexare proprie

Schema electronica din fig.3-16 utilizeaza un modul de afisare U3 cu trei elemente, cu catod comun, fiecare avand 7 segmente. Un circuit decodor U2 de tipul MMC22926 realizeaza functia de numarator si decodor. Butoanele de programare a tipului de numarare se interfateaza cu microcontrolerul pe doar doi pini , astfel fiind disponibili 7 linii IO pentru aplicatia utilizator. Microcontrolerul poate fi inlocuit cu succes cu PIC16F628 sau PIC12F7675

va duce la o modificare bruscă în afişarea valorii. Utilizatorul poate corecta această problemă prin software (nu se citeşte butonul decât o perioadă foarte scurtă, timp în care nu se generează CK), sau prin hardware, utilizând metoda derivativă de citire a butonului prezentată în 3.3.4. R13 are rolul de protecţie a pinului RB5 când acesta este setat ca ieşire în stare *high* şi butonul B5 este apăsat. Se remarcă rezerva **C**lock **O**ut (CO) a integratului 22926, care permite conectarea de tip înlănţuit (*daisy chain*) a mai multor astfel de dispozitive, practic fiind posibilă realizarea unui modul de afişare de până la 8-16 digiţi, depăşirea acestei valori ducând la întârzieri mari în transferul tactului spre dispozitivul cel mai îndepărtat de PIC. Limitarea curentului de segment este făcut automat de curentul de saturaţie al tranzistorului CMOS. Deşi metodele software de serializare vor fi prezentate într-un capitol viitor, aplicaţia de faţă utilizează cea mai simplă metodă JAL de serializare, instrucţiunea *for…loop…end loop* combinată cu generarea de tact. O variantă a bibliotecii 22926.jal este descrisă în continuare:

```
-- file     :  22925.jal
-- date     :  martie 2000
-- purpose  :  proceduri  pentru afişaj multiplexat de 4 digiţi MMC22925/MMC22926
-- includes:  fără
-- pins     :
--  b0 clock, (22926 pin 12)
--  b1 reset, (22926 pin 13)
--  b4 memorare, 22926 pin 5 şi pin 6 se conectează împreună pentru anihila
--  flicker-ul în modul numărare/afişare inversă (count down)
--  rolul header-ului de mai sus este esenţial pentru a înţelege ce ai făcut cândva demult…

var byte count = 0
var byte seconds = 0
var byte hundred = 0
var byte minutes = 0
var bit clock is pin_b0
var bit reset is pin_b1
var bit memo is pin_b4
pin_b0_direction = output
pin_b1_direction = output
pin_b4_direction = output  ; se setează ca intrare la momentul interogării butonului B5,
; în acest exemplu de program, B5 nu este utilizat de loc fiind la discreţia utilizatorului

procedure _22926_init is                    -- iniţializarea cipului
 clock = high
 reset = low
end procedure

procedure _22926_reset is                   -- ştergerea celor 4 digiţi
 reset = high
 asm nop
 reset = low
end procedure

procedure _22926_write ( byte in nr_1, byte in nr_2 ) is
 for nr_1 loop                   -- scrierea  nr_1 * nr_2 ; maxim 255 * 255
  for nr_2 loop
```

```
   clock = high
   clock = low
  end loop
 end loop
end procedure
procedure _22926_up ( byte in n ) is -- incrementarea a n < 255
 if count < n then
  count = count + 1
  clock = high
  clock = low
 end if
end procedure

-- incrementarea  a mai puţin de (cicle +1)*n , exemplu:
-- ( 3 + 1 ) * 250 = 1000      ( 9 + 1 ) * 100 = 10000
procedure _22926_full_up ( byte in cycle, byte in n )  is
 if hundred <= cycle  then
  if count == n then
   count = 0
   hundred = hundred + 1
  elsif count < n then
   count = count + 1
   clock = high
   clock = low
  end if
 end if
end procedure

-- decrementarea a n < 255 prin  resetare şi incrementare de  ( count - 1) tacţi
procedure _22926_down is
 if count > 0 then
  count = count - 1
  memo = low
  _22926_reset
   for count loop
    clock = high
    clock = low
   end loop
  memo = high
 end if
end procedure

-- decrementare zecimală (mai puţin de 1000 tacţi)
procedure _22926_full_down ( byte in n )is
 if hundred >= 1 then
   count = count - 1
   if count == 0 then
      count = n
      hundred = hundred - 1
   end if
   memo = low
   _22926_reset
    for count loop
    clock = high
```

```
    clock = low
   end loop
 _22926_write ( hundred, n )
 memo = high
 elsif hundred == 0 then
  _22926_down
 end if
end procedure
```

-- incrementare de ceas (secundele trebuiesc incrementate în programul principal)

```
procedure _22926_clock_up ( byte in minutes_max ) is
if minutes < minutes_max then
 if seconds == 60 then
      seconds = 0
      minutes = minutes + 1
 end if
memo = low
 _22926_reset
 for seconds loop
  clock = high
  clock = low
 end loop
_22926_write ( minutes, 100 )
memo = high
end if
end procedure
```

-- decrementare ceas, la 0.00 decrementarea se oprește, secundele trebuie incrementate în programul principal iar minutele trebuiesc definite înainte de utilizare

```
procedure _22926_clock_down is
 if ( minutes > 0 ) | ( seconds < 60 )  then
   if seconds == 60 then
      minutes = minutes - 1
      seconds = 0
   end if
   memo = low
   _22926_reset
    for ( 59 - seconds ) loop
     clock = high
     clock = low
    end loop
   _22926_write ( minutes, 100 )
   memo = high
 end if
end procedure
```

Aplicația care utilizează biblioteca prezentată, este un simplu program care setează în ambele moduri (zecimal şi ceas) afişajul în cauză, memorând starea anterioară astfel încât trecerea de la un mod de afişare la altul se face fără pierderea informației. Este un exemplu în care afişarea zecimală poate reprezenta o tensiune sau o frecvență iar timpul o setare de cronometru. Se utilizează o procedură descrisă anterior pentru generarea întârzierilor cu TMR0.

```
include 16f84_4
include jpic
include jdelay
include 22926
var byte save_count = 0
var byte save_hundred = 0
var byte save_seconds = 0
var byte save_minutes = 0
var bit flag_count_up = off, flag_count_down = off
var bit flag_clock_up = off, flag_clock_down = off
procedure tmr ( byte in prescaler_set, byte in tmr_set ) is
       clear_watchdog
       status_rp0 = on
       option_reg = prescaler_set
                              ;  aici este registrul fizic option_reg , bank1
       status_rp0 = off
       tmr0 = tmr_set
end procedure

 procedure point_on is ; aprinde punctul zecimal DP2 pe afişaj
  pin_b5_direction = output pin_b5 = low
 end procedure

 procedure point_off is ; stinge punctul zecimal şi foloseşte pin_b5 ca intrare
  pin_b5_direction = input
 end procedure

_22926_init
_22926_reset
tmr ( 7, 51 )
forever loop -- programul principal

pin_b3_direction = output          -- citeşte B1, decrementare zecimală
pin_b3 = low
pin_b2_direction = input
                       -- numai dacă pin_b2 şi t0if (la fiecare 50ms) trec în 0 logic
var bit but1 = pin_b2 | ( ! intcon_t0if )
 if ! but1 then flag_count_up = ! flag_count_up
  if flag_count_down then
     count = save_count
     hundred = save_hundred
     point_off                     -- punct zecimal stins – afişare zecimală
     _22926_full_up ( 3, 250 )
     save_hundred = hundred   save_count = count
  end if
 intcon_t0if = low
 end if

pin_b3 = high                      -- citeşte B4, decrementare cronometru
var bit but4 = pin_b2 | ( ! intcon_t0if )
 if ! but4 then flag_clock_up = ! flag_clock_up
  if flag_clock_down then
     seconds = save_seconds
```

```
      minutes = save_minutes
      point_on
      seconds = seconds + 1
      _22926_clock_up ( 9 )
      save_minutes = minutes
      save_seconds = seconds
   intcon_t0if = low
  end if

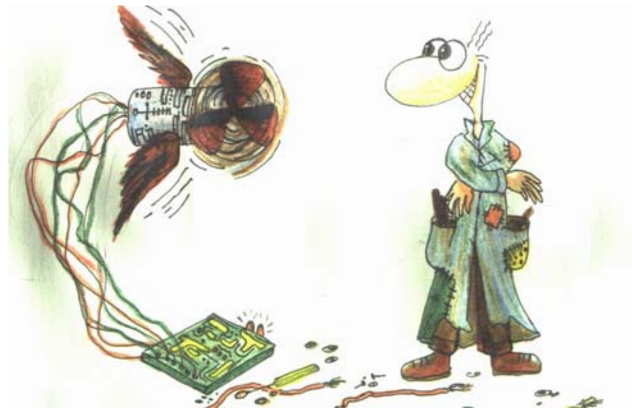
pin_b2_direction = output   -- citeşte B2, incrementare cronometru
pin_b2 = low
pin_b3_direction = input
var bit but2 = pin_b3 | ( ! intcon_t0if )
 if ! but2  then flag_clock_down = ! flag_clock_down
    if flag_clock_up then
       seconds = save_seconds
       minutes = save_minutes
       point_on                         ; punct activ doar pentru cronometru
       seconds = seconds + 1            ; incrementarea solicitată în biblioteca 22926.jal
       _22926_clock_down
       save_minutes = minutes
       save_seconds = seconds
   intcon_t0if = low
  end if

pin_b2 = high   -- citeşte B3, decrementare zecimală
var bit but3 = pin_b3 | ( ! intcon_t0if )
 if ! but3 then flag_full_down = ! flag_full_down
    if flag_count_up then
       count = save_count
       hundred = save_hundred
       point_off
       _22926_full_down ( 250 )
       save_hundred = hundred
       save_count = count
    end if
   intcon_t0if = low
  end if

end loop
```

O observaţie importantă este modul de salvare în regiştrii SRAM după fiecare operaţie de incrementare/decrementare, fie că este vorba de afişarea zecimală (0 - 999), fie de afişarea cronometrului (0 - 9.59). Dacă această salvare este omisă, presupunând că a avut loc anterior o setare a cronometrului şi s-a comutat pe afişare zecimală, valoarea afişată va fi ultima valoare incrementată în operaţia precedentă (adică o afişare de cronometru), ceea ce este evident incorect deoarece se aşteaptă afişarea ultimei valori a modului de lucru curent. O altă particularitate este şi utilizarea flagurilor în regim *toggle* adică negarea acestora după testare, cu rolul iniţializării viitoare a butonului opus (B1-B3 respectiv B2-B4) unde rolul butonul antagon decrementării este incrementarea (a nu se confunda cu tastele funcţionale, acest exemplu nu utilizează taste funcţionale; două butoane setează valoarea ceasului afişat iar alte două, valoarea zecimală afişată).

3.5 Interfațarea dispozitivelor inductive



Dintre dispozitivele inductive comune fac parte releele, difuzoarele, electroventile sau alte dispozitive de acționare cu solenoizi, motoarele de curent continuu, motoare cu acționare pas cu pas. Microcontrolerului i se potriveşte ca o mănuşă acționarea celor din urmă, datorită aspectului digital al comenzii care permite o interfațare mult mai uşoară decât în cazul motoarelor clasice, îndeosebi pentru obținerea unor regimuri de funcționare proporționale (reglarea simplă a turației şi a sensului de învârtire al motoarelor pas cu pas la momente precise de timp). Practic motorul pas cu pas poate fi considerat un convertor proporțional digital-mişcare (şi cu puțină imaginație şi revers, cu funcția de encoder digital).

Clasificarea motoarelor pas cu pas	
După principiu de funcționare:	După modul de alimentare al bobinelor:
cu magnet permanent	unipolare
cu reluctanță variabilă	bipolare
hibride	

După numărul de bobine ce trebuie comandate există motoare cu două, trei şi patru secțiuni de bobine. Cele cu mai mult de patru au destinatii speciale şi sunt rar utilizate. Din punctul de vedere al rezoluției, cele mai comune motoare au: 15, 7.5, 3.75, 3.6, 1.8 şi 0.9 ^{O}/pas, însă toate permit comanda în regim *half-step* (jumătate de pas) în timp ce utilizarea unor drivere specializate asigură comanda motoarelor pas cu pas în regim *micro-step*. Unghiul de rotație pentru un pas al unui motor pas cu pas cu magnet permanent este determinat de relația existentă între numărul de poli ai statorului (electromagnet) şi numărul de poli ai rotorului (magnet permanent). Ultimul are un număr redus de poli ce depinde de geometria magnetului şi caracteristicile materialului magnetic din care este confecționat. Creşterea rezoluției este posibilă prin utilizarea unui "sandwich magnetic" pe rotor şi decalarea polilor magnetici între feliile "sandwich-ului". Astfel practic se dublează rezoluția printr-un artificiu mecanic. Un motor pas cu pas se identifică după câteva aspecte:

- învârtind axul rotorului va simți cuplul generat de câmpul magnetic al rotorului aflat în interacție cu magnetizarea remanentă a statorului, ca o reacție digitală discontinuă în paşi, alternând un cuplu puternic cu unul slab, ambele fiind mai apropiate cu cât rezoluția motorului este mai fină.
- după numărul de fire ce ies din rotor, motoarele unipolare au 5 sau 6 fire, uneori conexiunile comune având culorile combinate ale terminalelor corespunzătoare de pe capetele opuse (roşu spiralat cu negru este terminalul comun pentru ambele bobine terminate cu fire de culoare roşie respectiv neagră); motoarele bipolare au doar 4 fire.
- după eticheta ce precizează tensiunea de alimentare şi numărul de grade/pas

Ceea ce este puțin mai dificil şi necesită răbdare, este determinarea ordinii de conectare a înfăşurărilor pentru a obține sensul de învârtire dorit. Se utilizează o sursă de alimentare şi se conectează (la motoare unipolare), conexiunea comună la +V iar celelalte înfăşurări pe rând la –V, până când sensul de învârtire este corect, fie înainte fie înapoi, conform tabelelor din 3.5.1. In cazul motoarelor bipolare este ceva mai dificil, deoarece trebuiesc schimbate polaritățile de alimentare ale celor două bobine până la obținerea unui sens corect de învârtire, fără pierdere de paşi sau întoarcere înapoi, conform tabelului din 3.5.3.

3.5.1 Motoare pas cu pas unipolare

Motoarele pas cu pas unipolare sunt cele mai simplu de comandat şi utilizat deoarece nu necesită schimbarea sensului curentului prin bobinele de comandă ci doar comutarea acestora. Fig.3-17 prezintă principul de functionare al unui motor de acest tip, cu patru faze. Imaginea a) indică situația când bobinele P şi R sunt alimentate prin intermediul comutatoarelor electronice S1,S2. Rotorul se va alinia pe direcția NS a polilor creați (polii

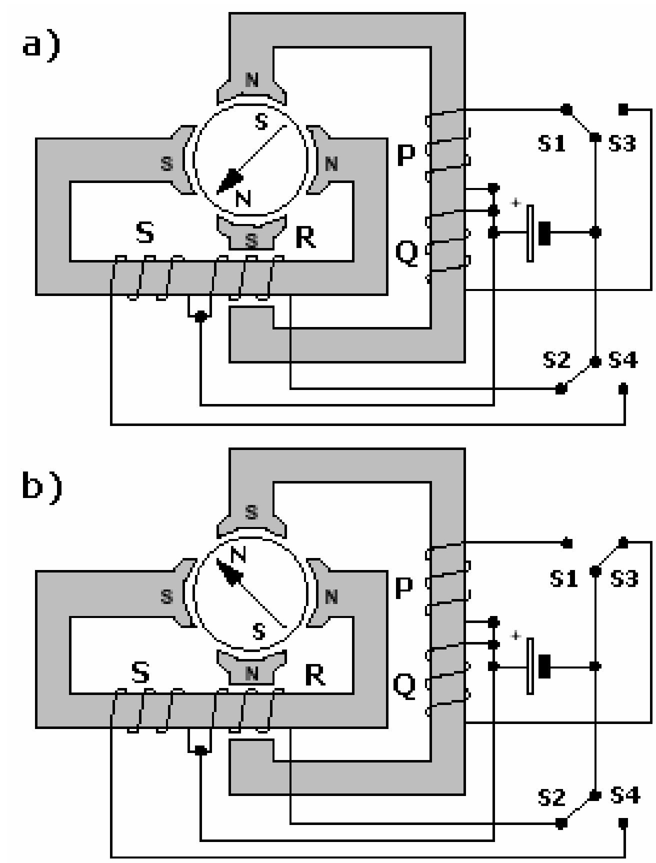


Fig.3-17 Principiul de funcționare al motorului pas cu pas unipolar

de acelaşi nume se resping iar cei cu nume contrar se atrag). Dacă se comută alimentarea bobinelor Q şi R prin S3 şi S2 ca în imaginea b), câmpul magnetic al statorului se reorientează învârtind rotorul încă un pas. Prin acționarea cu viteză convenabilă şi în

succesiunea dorită a comutatoarelor S1, S2, S3 şi S4, se poate obţine mişcarea rotorului în sensul şi cu viteza dorită. Cititorul a întâlnit deja acest tip de motor în cap. 2.11.3. Aici driverul utilizat era ULN2803 (CD:\datasheet\texas\ULN2803.pdf). Acesta poate comanda două astfel de motoare. Deşi acest circuit integrat este relativ ieftin şi uşor de obţinut, se poate imagina o schemă utilizând tranzistoare discrete, mai ales pentru curenţi ce depăşesc 0.5…2A, curenţi pe care driverul prezentat nu îi poate comanda. Deşi fig.3-18 prezintă o instalaţie relativ complexă de călire a unor piese mici în flacără de gaz, pe care am realizat-o acum câţiva ani şi care funcţionează şi azi, o inspecţie sistematică a schemei va evidenţia elementele aflate în discuţie. Motorul pas cu pas unipolar M1, are tensiunea nominală de 5V şi un curent nominal de vârf de cca. 2 A. Se observă că motorul are 6 fire de conexiune din care terminalele comune au fost conectate în paralel la borna 5 a conectorului J2, spre +5V. Pentru a evita resetările parazite ale microcontrolerului în momentul comenzii motorului, este imperativă conectarea unui condensator de deparazitare de 220uF…1000uF în imediata apropriere a conexiunii comune a motorului. Dacă motorul se amplasează la mare distanţă de placa de comandă (peste 1 m), tranzistorii de comandă Q1…Q4 trebuie să se găsească în imediata apropriere a motorului pe o placă de circuit imprimat separată de cea a microcontrolerului, comanda acestora supunându-se regulilor de transfer a semnalelor logice la mare distanţă. Esenţiale sunt de asemenea diodele supresoare de stingere a oscilaţiilor datorate inductanţelor bobinelor, D1…D4. Aceste diode trebuiesc montate deasemenea în apropierea bobinelor motorului pentru a micşora cât de mult lungimea circuitului de supresie care se comportă ca o antenă generatoare de zgomot. Comanda tranzistoarelor drivere este realizată din portul A al PIC-ului, prin rezistenţe de limitare a curentului de bază de 3k3. Secvenţa logică de comandă cu pas întreg şi pas pe jumătate a acestui tip de motor, se poate vedea în tabelul următor. Numele bobinei este echivalent cu numele terminalului din cupla J2 (J2-1 este echivalent cu Bobina1):

ROTIRE INAINTE FULL STEP-standard				**ROTIRE INAPOI FULL STEP-standard**			
Bobina1	Bobina 2	Bobina3	Bobina4	Bobina1	Bobina 2	Bobina3	Bobina4
+5V	0V	0V	0V	0V	0V	0V	+5V
0V	+5V	0V	0V	0V	0V	+5V	0V
0V	0V	+5V	0V	0V	+5V	0V	0V
0V	0V	0V	+5V	+5V	0V	0V	0V

Este esenţială în acest moment, cunoaşterea a două mărimi ce definesc mişcarea motorului:

- Cuplu de menţinere (Nm): este cuplul maxim ce poate fi aplicat pe axul unui motor alimentat, fără a cauza o mişcare de rotaţie
- Cuplu maxim de lucru (Nm): este cuplul maxim ce poate fi obţinut la axul motorului în funcţionare.

ROTIRE INAINTE FULL STEP-power				**ROTIRE INAPOI FULL STEP-power**			
Bobina1	Bobina 2	Bobina3	Bobina4	Bobina1	Bobina 2	Bobina3	Bobina4
+5V	+5V	0V	0V	0V	0V	+5V	+5V
0V	+5V	+5V	0V	0V	+5V	+5V	0V
0V	0V	+5V	+5V	+5V	+5V	0V	0V
+5V	0V	0V	+5V	+5V	0V	0V	+5V

Pentru situația descrisă de tabelul **full step-standard**, atât cuplul de menținere cât şi cuplul de lucru sunt reduse cu 30% față de modul **full step-power** sau **half step-power**. Singurul mod de compensare al acestui neajuns, este creşterea tensiunii de alimentare în anumite limite rezonabile.

ROTIRE INAINTE HALF STEP-power				**ROTIRE INAPOI HALF STEP-power**			
Bobina1	Bobina 2	Bobina3	Bobina4	Bobina1	Bobina 2	Bobina3	Bobina4
0V	0V	+5V	+5V	+5V	+5V	0V	0V
0V	0V	+5V	0V	0V	+5V	0V	0V
0V	+5V	+5V	0V	0V	+5V	+5V	0V
0V	+5V	0V	0V	0V	0V	+5V	0V
+5V	+5V	0V	0V	0V	0V	+5V	+5V
+5V	0V	0V	0V	0V	0V	0V	+5V
+5V	0V	0V	+5V	+5V	0V	0V	+5V
0V	0V	0V	+5V	+5V	0V	0V	0V

In modul **half-step-power**, viteza de rotație a motorului scade la jumătate, iar rezoluția la deplasare creşte de două ori. Pentru un motor având 1.8^{O}/pas care funcționează în regimul **half-step** rezoluția obținută va fi 0.9^{O}/pas, suficientă în majoritatea aplicațiilor curente. Scrierea programului de comandă este mult simplificat de existența bibliotecii jstepperm.jal care conține toate rutinele de lucru (full-step standard, half-step şi power-step):

```
procedura învârte_înainte is ( byte in viteza ) ; viteza = 5…25
stepper_motor_half_forward( stepper )
     port_a = stepper
     delay_1ms( viteza )
end procedure
procedura învârte_înapoi is
stepper_motor_full_backward( stepper )
port_a = stepper
     delay_1ms( 10 )
end procedure
```

Din cauza apelării înlănțuite a procedurii din procedură (procedura apelată în programul principal este învârte_înainte sau învârte_înapoi, iar aceasta apelează procedura stepper_motor_x_x), acest mod de apelare consumă două locații din stivă ceea ce ar putea crea probleme programelor foarte lungi prin consumarea prematură a stivei microcontrolerului. Soluția posibilă este apelarea în linie direct în programul principal a rutinelor de motor, renunțând la rutinele intermediare (folositoare de altfel numai pentru înțelegerea ulterioară a programului).

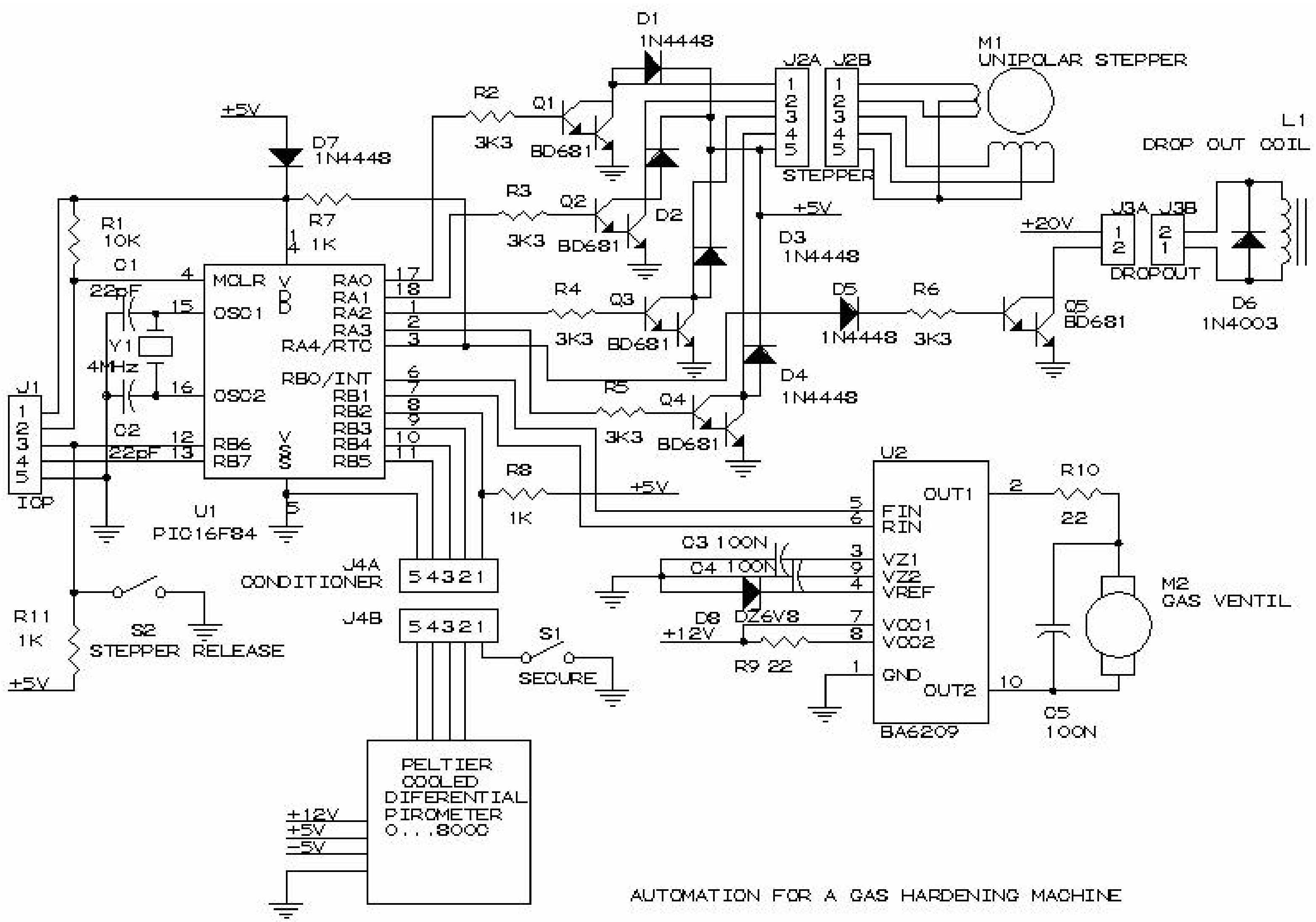


Fig.3-18 Exemplu de comandă a dispozitivelor inductive comune

Schema electronica reprezinta un dispozitiv de calire in flacara de gaz care comanda multiple dispozitive electromagnetice: motorul pas cu pas M1, motorul de curent continuu cu perii si stator cu motor permanent M2, electromagnetul de aruncare L1.

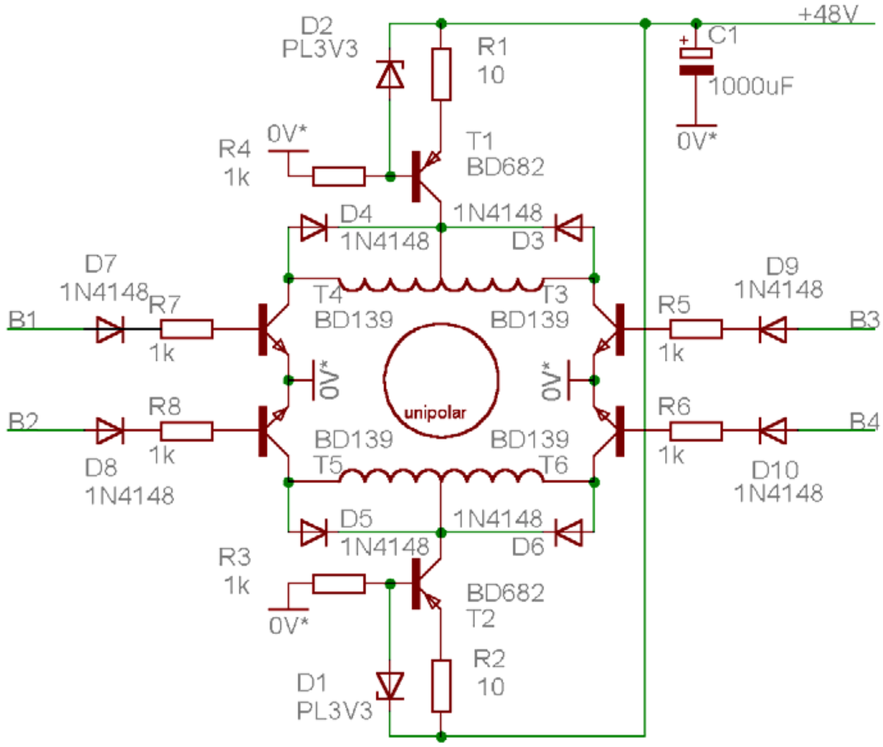


Fig.3-19 Alimentarea motorului unipolar sub curent constant

Tensiunile nominale standardizate pentru motoarele pas cu pas de mică putere sunt 5V, 12V, 24V şi 48V. Nu sunt probleme deosebite pentru viteze mici de rotație a acestor motoare. Problemele încep când este necesară funcționarea acestora la frecvențe mai mari de 1KHz. Se utilizează artificii hardware de creştere a vitezei de variație a tensiunii pe bobine, prin utilizare alimentării sub curent constant, de la tensiuni de alimentare de 3…5 ori mai mari decât tensiunile nominale. O schemă tipică pentru frecvențe de comandă ridicate şi motoare pas cu pas unipolare este exemplificată în fig.3-19, tensiunea nominală a motorului fiind 12V. Se observă că bobinele sunt alimentate din două generatoare identice de curent constant format din T1, T2 şi componentele aferente. Diodele zener D1 şi D2 asigură referința de 2.1V (3.3V – 1.2V căderea de potențial pe joncțiunea bază-emitor a tranzistorului darlington T1,T2) pe rezistențele de sesizare R2 şi R1. Astfel, curentul maxim generat de acestea este de cca. 210mA. Puterea disipată de tranzistoarele T1 şi T2 este în momentul comenzii, de (48V – 12V) x 0.21A = 7.3VA iar puterea disipată de diodele D1 şi D2 este de aproximativ 150mW. Este obligatorie utilizarea unui radiator dimensionat corespunzător pentru aceste tranzistoare. Cheia funcționării cu viteze ridicate (deci cu impulsuri de comandă foarte scurte) este flotarea sarcinii atât față de masă, prin tranzistoarele de comandă cât şi față de tensiunea de alimentare prin generatoarele de curent, viteza de variație a tensiunii pe bobine fiind dependentă şi de impedanța echivalentă de comandă văzută de bobină. Obținerea unor frecvențe de comuntație mai mari de 2KHz nu se poate face decât utilizând tranzistori de comutație adecvați (T3,T4,T5,T6) separarea alimentării sub curent constant a fiecărei din cele 4 bobine împreună cu renunțarea la

suprimarea directă a componentei inductive prin diodele D3, D4, D5 şi D6, în favoarea protejării joncţiunilor CE a tranzistorilor de comutaţie şi a generatoarelor de curent constat.

3.5.2 Relee şi solenoizi

Cum se comandă un solenoid flotat faţă de masă, (având unul dintre terminale conectat la o tensiune mult mai mare decât +5V) se poate vedea tot în fig.3-18. Este nevoie de aceleaşi precauţii de supresare a componentei inductive a tensiunii de comutaţie (prin dioda D6) şi precauţii suplimentare în circuitul de comandă (D5 şi R6) pentru situaţia de defect prin străpungere parţială sau totală a joncţiunii bază-colector a tranzistorului Q5. Un aspect interesant care apare în practică, este comutaţia parazită a releelor sau solenoizilor a căror comandă de acţionare este pe nivel logic *high,* deoarece până la setarea direcţiei şi a comenzii pinului de ieşire al PIC-ului destinat comenzii solenoidului, regimul tranzitoriu de pornire al microcontrolerului va genera un puls de scurtă durată *high-low*. Pentru a micşora acest puls cât de mult posibil, este necesar ca definirea pinilor care comandă relee să fie făcută imediat la începutul secvenţei de program şi nu după ce mai multe rutine consumatoare de timp au fost executate. Pentru exemplul nostru:

```
include 16f84
include jpic

var volatile bit relay is pin a4
pin_a4_direction = output
relay = low
```

Dacă nu se poate înlătura comutarea parazită în acest mod, este necesară utilizarea unei comenzi integrative (dacă aplicaţia o permite), integrarea făcându-se cu o constantă de timp ceva mai mare decât maximul pulsului parazit apărut la alimentarea PIC-ului. Astfel şi comenzile efectuate în mod curent vor avea o întârziere din momentul comenzii şi până în momentul execuţiei, însă ţinând cont că cel mai bun releu are o constantă de răspuns de 2…8 mS, adăugarea unei milisecunde în plus nu afectează rezultatul comenzii.

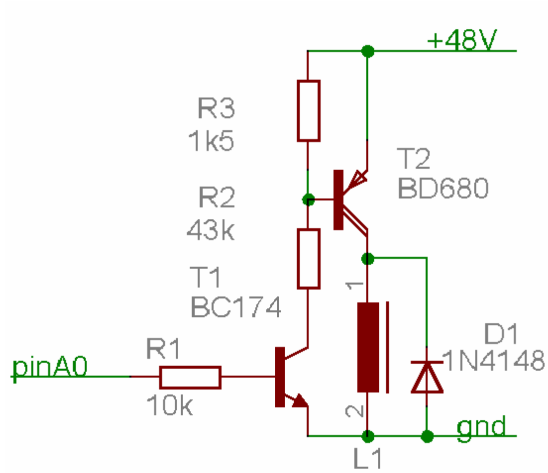


Fig.3-20 Comanda unui solenoid cu tensiune ridicată, faţă de masă

Există şi situaţia când este absolut necesară comanda unui solenoid către masa circuitului

iar solenoidul are tensiunea nominală mult mai mare decât tensiunea de alimentare a microcontrolerului. Singura precauţie este utilizarea unor tranzistoare care să suporte tensiunile şi curenţii necesari comenzii respective. Filtrajul suplimentar cât mai aproape de bobina solenoidului este deasemenea recomandat. Când sunt necesari curenţi tari, este obligatorie separarea masei digitale a microcontrolerului de masa de forţă a solenoidului şi conectarea acestora într-un punct cu impedanţă minimă din imediata apropiere a sursei de alimentare. Dimensionarea reţelei R2, R3 se face ţinând cont de curentul minim de comandă al T2 şi tensiunii BE a acestuia de 1.2V (darlington).

Nu întotdeauna este suficientă comanda monopolară (on-off) a unui solenoid (fig.3-21). Există situaţii când solenoidul necesită alimentare bipolară. Aplicarea unei polarităţi pe bobina solenoidului va deschide, de exemplu, o cale al distribuitorului de gaz iar schimbarea polarităţii o va închide şi va deschide o altă cale. Lipsa tensiunii de alimentare va bloca ambele căi de acces. Metoda poartă denumirea de comandă în semipunte, şi foloseşte ideea din fig.3-20 atât pentru polarităţi pozitive cât şi pentru polarităţi negative ale sarcinii, păstrîndu-se referinţa de comandă faţă de masă. Este evident că în locul solenoidului se poate găsi orice fel de sarcină, de la una rezistivă şi până la motoare de curent continuu cu perii. Grupul T1, T2 funcţionează identic cu montajul prezentat în fig.3-20, în timp ce T3, T4, T5 realizează comanda cu tensiune negativă. A fost necesară această decalare de nivel a comenzii spre +48V (cu T5) respectiv spre –48V (cu T4), deoarece microcontrolerul are comenzile de ieşire A respectiv B raportate la masa solenoidului. Presupunând că A este în stare logică *high*, T1 este în conducţie asigurând polarizarea lui T2 prin R8 şi generarea unui curent pozitiv prin solenoid faţă de masă.

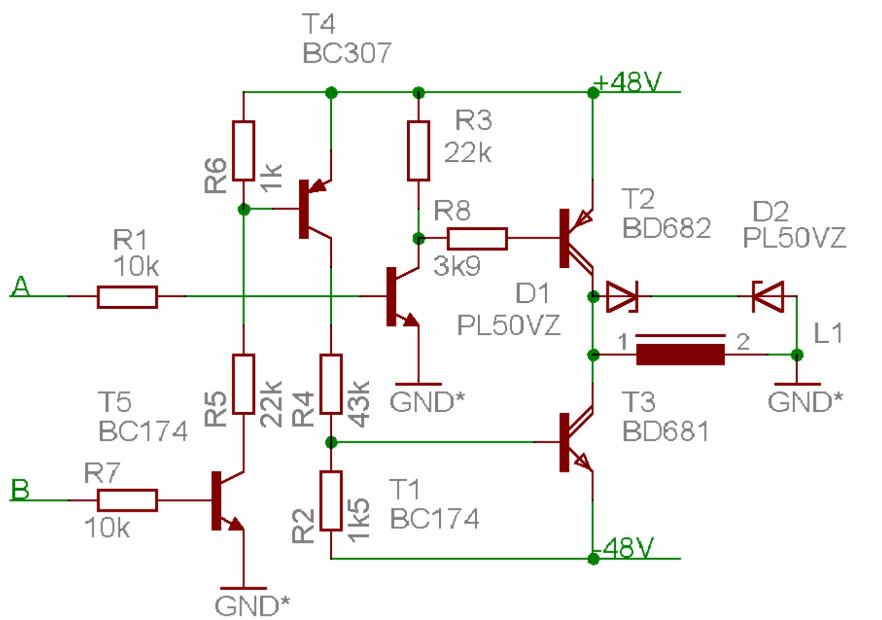


Fig.3-21 Comandă în semipunte

In situaţia când B trece în *high*, asigură polarizarea tranzistorului T4, căderea de tensiune pe R2 va fi suficient de mare pentru a deschide tranzistorul T3 care va extrage un curent prin solenoidul L1 dinspre masă înspre –48V. Deoarece sensul curentului prin bobină se schimbă, diodele supresoare D1, D2 trebuie să asigure conducţia pentru ambele sensuri, fiind conectate astfel încât întotdeauna una din diode se comportă ca o diodă obişnuită în timp ce, dioda opusă are rol de diodă zener. Valoarea tensiunii diodelor se alege ceva mai

mare decât tensiunea de alimentare, pentru a preîntâmpina conducţia lor forţată urmată de scurcircuitarea lor. Este evident că o comandă greşită (atât pe A cât şi pe B în acelaşi timp) duce la distrugerea tranzistoarelor T2 şi T3 dacă sursele de alimentare nu au protecţie la scurtcircuit. Pentru situaţia când ambele comenzi au nivel logic *low*, menţinerea în stare blocată a tranzistoarelor T2 şi T3 se face de către grupul R3, R8 respectiv R2. Bineînţeles că pentru aceasta, ambele tranzistoare T1 şi T5 trebuie să fie blocate. Tensiunile de alimentare (±48V) provin din circuite de redresare şi filtrare standard.

3.5.3 Motoare pas cu pas bipolare

Motoarele pas cu pas bipolare au doar două bobine. Pot avea aceleaşi rezoluţii ca motoarele unipolare însă necesită circuite de comandă mult mai elaborate. Avantajul acestora de a avea un cuplu cu peste 30% mai mare decât motoarele unipolare, pentru turaţii

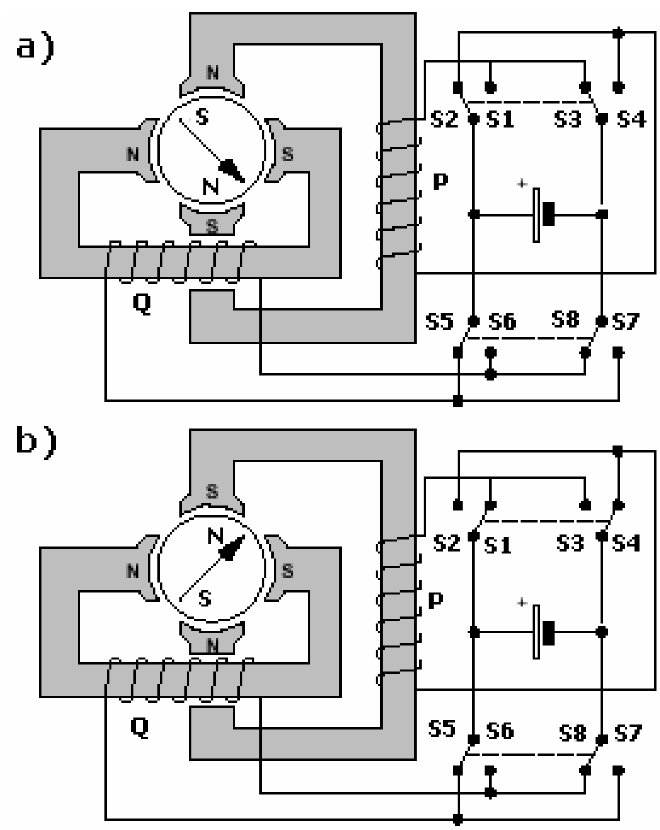


Fig.3-22 Secţiune printr-un motor pas cu pas bipolar

mai mici de 100 de paşi/secundă, compensează complexitatea driverului necesar.
Fig.3-22 a) şi b) explică funcţionarea motorului prin schimbarea polarităţii curentului prin bobine. Comutarea se realizează întotdeauna cu comanda perechilor de comutatoare (drivere) S1-S4; S6-S7 respectiv S2-S3; S5-S8. Spre deosebire de motoarele unipolare care dispun de patru bobine într-un spaţiu fizic identic ca dimensiune cu cel al motoarele bipolare, grosimea sârmei şi numărul de spire al celor două bobine ale motoarelor bipolare este mai mare, cu efect benefic pentru puterea extrasa la axul motorului. Motorul bipolar

necesită schimbarea sensului curentului prin bobine pentru a obţine câmpul magnetic învârtitor. Din această cauză necesită fie comandă duală în semipunte, fie comandă duală în punte.

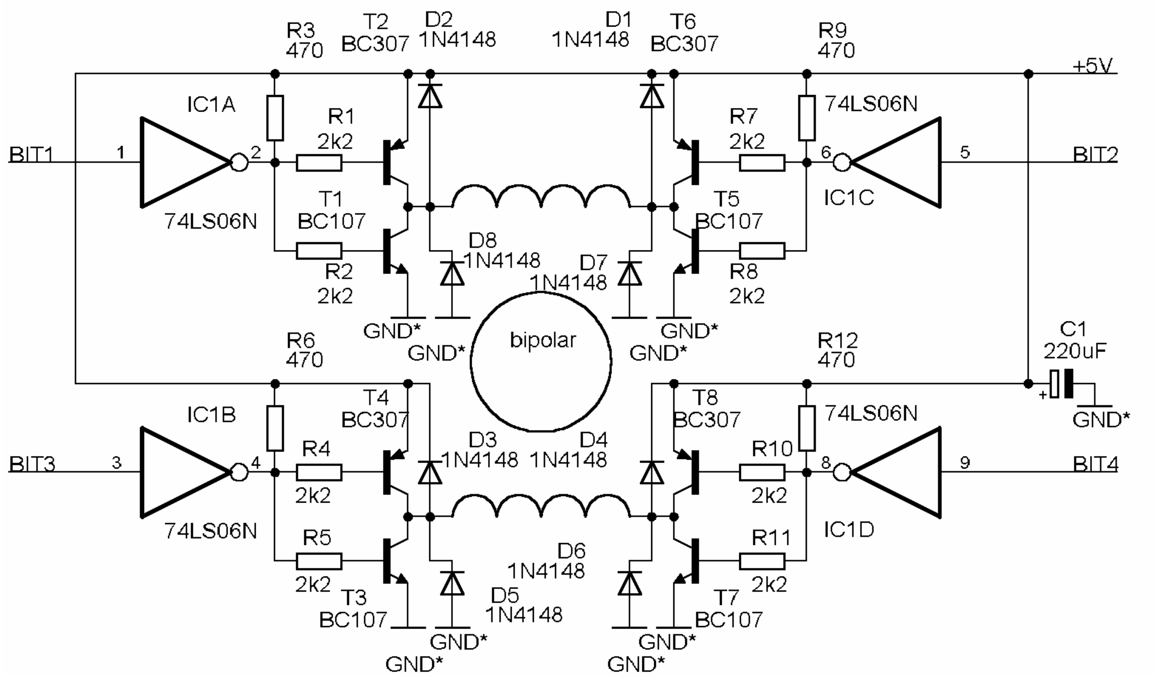


Fig.3-23 Comanda unui motor bipolar (de avans) recuperat din unitatea de floppy-disk

Deşi aceste etaje de comandă pot fi proiectate cu componente discrete, se preferă utilizarea circuitelor de comandă specializate. Fig. 3.23 şi tabelul următor evidenţiază cum se poate realiza foarte simplu comanda unui motor pas cu pas bipolar de avans a capului magnetic, ce echipează orice unitate de floppy-disk de 3.5 inch,:

ROTIRE INAINTE FULL STEP-bipolar				ROTIRE INAPOI FULL STEP-bipolar			
Bit1	Bit2	Bit3	Bit4	Bit1	Bit2	Bit3	Bit4
1	0	X	X	X	X	0	1
X	X	1	0	0	1	X	X
0	1	X	X	X	X	1	0
X	X	0	1	1	0	X	X

Microcontrolerul se interfaţează prin patru bufere inversoare open colector 74SN06. Se pot utiliza atât bufere inversoare cât şi neinversoare (405, 406, 407, 417) cu modificarea corespunzătoare a secvenţei de comandă. Urmărind tabelul de comandă pentru rotaţia cu pas întreg, se poate analiza funcţionarea punţii de alimentare a uneia dintre bobinele motorului. Presupunând că **bit1** este în stare logică *high* şi **bit2** în stare logică *low,* vor intra în conducţie T2 respectiv T5, curentul prin bobina corespunzătoare a motorului având sensul convenţional indicat de săgeata emitorilor tranzistorilor T2 şi T5. O schimbare a comenzilor va modifica sensul curentului prin bobina motorului pas cu pas. Tranzistoarele de comandă sunt de tip comun de 100mW, însă la viteze mai mari se recomandă utilizarea

tranzistoarelor de comutație cu tensiune mică de saturație colector emitor, cum ar fi 2N706 şi 2N2369. Se observă diodele supresoare a inducției produse în bobine, conectate să conducă spre potențialele cu impedanță minimă (în cazul de față sursele de alimentare). Dezavantajul major al schemei este numărul mare de componente necesar în cazul alimentării unipolare de +5V. Dacă sunt accesibile tensiuni bipolare de ±5V, se pot utiliza două semipunți, comanda bobinelor făcându-se spre masă şi astfel se reduce numărul de tranzistoare de la 8 la 4. Circuite integrate dedicate comenzii motoarelor pas cu pas bipolare sunt: TEA3717, poate comanda tensiuni de 10…45V şi curenți de ±1A; UC3173 funcționează la 5V sau 12V şi poate comanda ±500mA; UC3770 funcționează cu tensiuni cuprinse între 10…50V şi comandă curenți de ±2A. L3262E conține trei semipunți capabile să comande curenți de 1.5A atât în mod PWM cât şi liniar. Toate aceste circuite beneficiază de comandă pentru microstep şi protecție la scurtcircuit.

3.5.4 Interfațarea motoarelor de curent continuu

Comanda motoarelor de curent continuu cu perii se poate face în mod proporțional sau în mod tot-nimic (*on-off*) ambele cu sau fără schimbarea sensului de învârtire. Precauțiile luate la comutația solenoizilor se mențin, însă problema generării de paraziți este cu mult mai mare datorită ansamblului perii-colector, care se ştie că sunt un generator perfect de zgomot datorită scânteilor ce apar la sarcini mari, bătăi ale sarcinii mecanice la axul motorului sau la contacte imperfecte la nivelul colectorului. Comanda on-off cu schimbarea sensului, se face prin aceeaşi clasică punte de comandă. In fig.3-18 motorul de curent continuu M2, este alimentat prin intermediul circuitului integrat specializat BA6902, o punte de comandă generând curenți de până la 300mA, având comenzi digitale, interfațabile cu microcontrolerul, numite *forward* (înainte) şi *reverse* (înapoi). Circuitul integrat este ieftin (sub 1 euro) şi se găseşte în două tipuri de capsule: **S**ingle **I**n **L**ine cu 10 pini, din care cea de curent mare are un "steguleț" de cca 2cm^2 uşor montabil pe radiator. Schema de principiu a acestui circuit integrat (fig.3-24) evidențiază o logică de control,

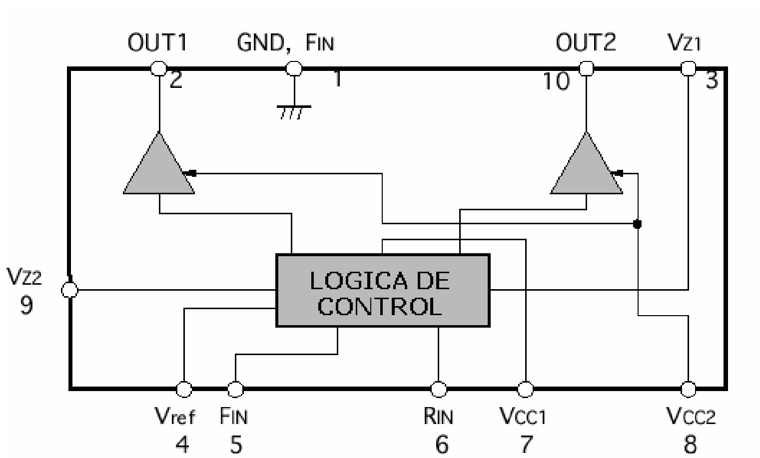


Fig.3-24 Schema bloc a circuitului integrat BA6209 produs de Rohm

două amplificatoare de ieşire cu tensiune de alimentare separată de blocul de control, o intrare de referinţă şi două ieşiri cu tensiuni obţinute prin stabilizare parametrică cu diode zenner. Schema tipică de aplicaţie pentru acest circuit integrat este prezentată în fig. 3-25.
Motorul din figură are tensiunea de alimentare de 6V şi un curent nominal de 100mA pentru capsula SIL10 fără radiator, respectiv maximum 300mA pentru capsula SIL10 cu radiator. Atât filtrajele referinţelor cât mai ales filtrajul zgomotului indus de perii (C1) este foarte important. Se recomandă de asemenea înserierea unei rezistenţe Rc de limitare a curentului prin motor în cazul blocării acestuia sau funcţionării în depăşire de sarcină. Comenzile sunt Rin (*reverse*) respectiv Fin (*forward*). Este un circuit integrat perfect pentru aplicaţii de comandă a motoarelor de curent continuu în modele radiocomandate. Dacă se doreşte obţinerea unei comenzi proporţionale, tensiunea de alimentare trebuie să fie variabilă. Metoda cea mai simplă pentru obţinerea acesteia este generarea unui semnal ***P**uls **W**ith **M**odulation* din microcontroler, (fig. 3-26) urmată de filtrarea acestuia.

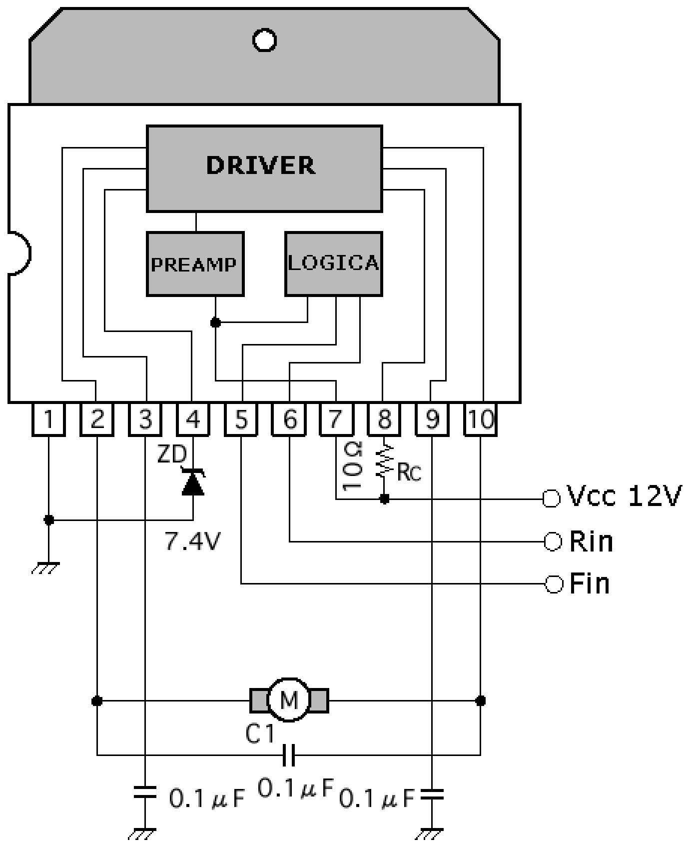


Fig.3-25 Schema tipică de aplicaţie BA6209

Programul ce comandă acest driver este extrem de simplu:

```
var bit forward is pin_b0
pin_b0_direction = output
var bit reverse is pin_b1
pin_b1_direction = output
procedure reverse is
  forward = low  reverse = high end procedure
procedure forward is
  forward = high reverse = low  end procedure
procedure stop is forward = low reverse = low end procedure
```

Se observă că procedurile sunt simetrice în funcţie de polaritatea de conectare a motorului, comenzile *reverse* şi *forward* pot fi reciproce. In funcţie de perioada de timp cât sunt activate procedurile, se pot obţine efecte de comenzi proporţionale prin integrarea mişcării mecanice. Dacă dispozitivul comandat are un volant, sau frecvenţa PWM este suficient de ridicată, efectul de "smucitură" la comutarea tensiunii de alimentare se diminuează considerabil. Programul următor realizează o învârtire a motorului cu turaţie redusă la jumătate faţă de situaţia continuă de alimentare obţinută prin rularea la nesfârşit a

procedurilor *forward* sau *reverse*, schimbând şi sensul de învârtire după aproximativ 4 secunde. Schimbarea factorului de umplere (valoarea variabilei *pulse*) cu păstrarea constantă a frecvenței sau periodei de repetiție (suma celor două variabile pentru forward + stop sau reverse + stop trebuie să rămână o constantă, aici 255) ne-a condus la definiția semnalului PWM de care aminteam mai sus. Este un exemplu de generare PWM prin software.

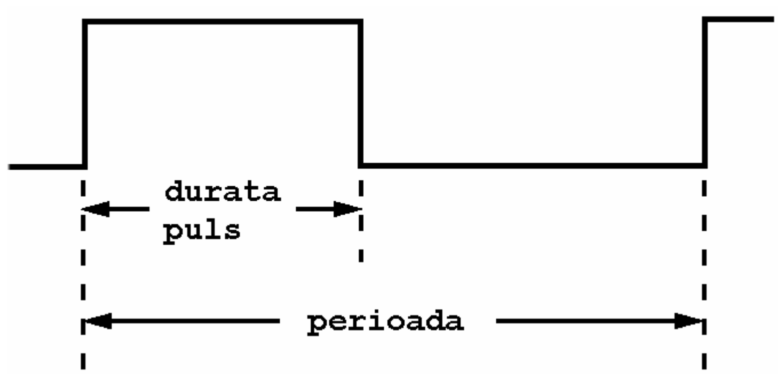


fig.3-26 Definirea semnalului PWM

```
var byte pulse = 128
for 100 loop
  forward delay_100uS ( pulse )
  stop delay_100uS ( 255-pulse )
end loop
for 100 loop
  reverse delay_100uS ( pulse )
  stop delay_100uS ( 255-pulse )
end loop
```

Semnalul PWM standard are ca parametri: durata (*period*) sau frecvența (f = 1/durata), lărgimea pulsului (*pulse with*) şi factorul de umplere (*duty cycle*).

$$factor_de_umplere = \frac{durata_pulsului}{perioada_semnalului} \times 100\%$$

unde

$$perioada_semnalului = 1/frecventa$$

Există o soluție analogică integrativă de obținere a unei tensiuni continue dintr-un semnal PWM. Ideea este de a utiliza un integrator RC având constanta de timp de cel puțin 3x până la 10x mai mare decât perioada semnalului PWM. Metoda merge pentru situația când nu suntem nevoiți să schimbăm frecvența PWM la intervale definite de timp. Dacă ieşirea microcontrolerului are etajul de ieşire perfect simetric dpdv al tensiunii de saturație, pentru ambele tranzistoare MOS ce încarcă (T2) şi respectiv descarcă (T1) condensatorul, va rezulta un semnal continuu proporțional cu factorul de umplere al PWM. Erori sunt posibile pentru un factor de umplere foarte mic când semnalul de ieşire este aproape de zero şi

descărcarea nu se face complet datorită "saturaţiei" tranzistorului MOS, $R_{DS}T1 > 0$ (fig.3-27)

$$T = R_1 \times C_1 = 3\ldots10 > 1/F_{pwm}$$

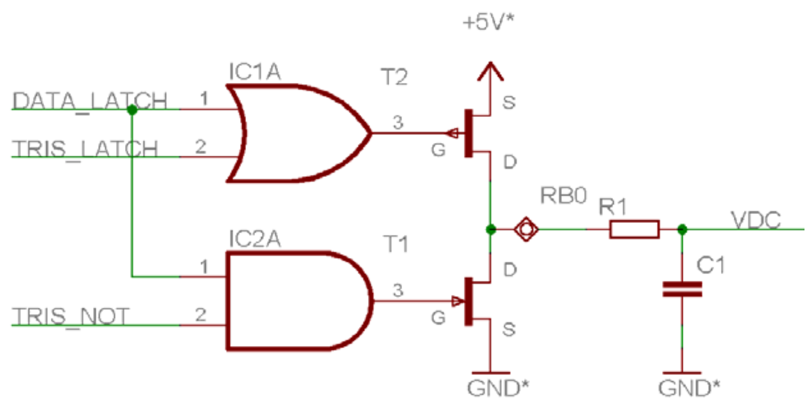


Fig.3-27 Structura internă a unui pin al PIC configurat ca ieşire PWM

Dimensionarea reţelei R1, C1 se face alegând convenabil o valoare pentru capacitatea C1 şi apoi calculând valoarea rezistenţei R1. Un neajuns important al acestui mod de filtrare este impedanţa mare de intrare necesară circuitului deservit de această tensiune continuă, proporţională cu factorul de umplere al semnalului PWM. Utilizarea unui repetor cu amplificator operaţional pentru tensiunea VDC este absolut necesară când impedanţa sarcinii este mai mică sau egală cu R1. Constanta de filtrare (3T…10T) are valori mici când riplul VDC nu este important dar viteza de variaţie a semnalului de ieşire este esenţială, respectiv valori mari când semnalul poate fi lent variabil dar trebuie să fie filtrat perfect. Este posibilă şi utilizarea unui circuit integrator cu amplificator operaţional. Ideea descrisă anterior este cea mai simplă metodă de conversie DA cu microcontroler.

3.5.5 Interfaţarea motoarelor cu reluctanţă variabilă

Motoarele cu reluctanţă variabilă sunt foarte asemănătoare cu motoarele pas cu pas unipolare, având însă viteze de rotaţie mult mai mari şi trei (uneori şase) bobine de comandă. Se poate recunoaşte uşor după dimensiune, numărul de fire şi cuplul mecanic remanent datorat interacţiunii dintre rotor (un inel circular din oţel magnetizat) şi stator (un număr de bobine montate pe o placă de circuit imprimat). Este vorba şi despre unele motoare care învârt discheta în unitatea de 3 inch. Deşi motoarele au doar patru terminale active (3 active şi un terminal comun), sunt motoare cu reluctanţă variabilă care au până la 9 terminale din care o parte se utilizează ca feedback pentru reglarea turaţiei utilizând senzori hall. Senzorul hall generează un semnal dreptunghiular la fiecare trecere a polilor magnetici ai rotorului prin dreptul său. Driverele moderne pentru aceste motoare nu mai au nevoie de artificii de comandă utilizând însăşi bobinele motorului ca senzor de turaţie în perioada în care acestea nu sunt alimentate. Secvenţa de comandă este identică cu cea a motorului pas cu pas unipolar în configuraţia standard de comandă, aplicată însă pentru doar trei înfăşurări. Un circuit integrat specializat pentru comanda acestui tip de motor este TDA 5142 de la Philips, el comandă un curent de max. 200mA.

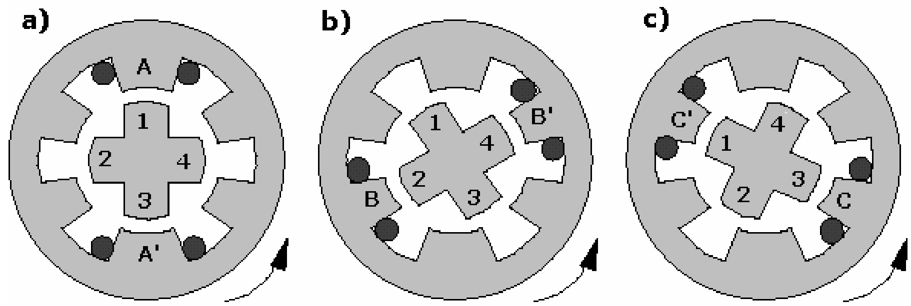


Fig.3-28 Motor cu reluctanţă variabilă

Aşa cum arată fig.3-28, rotaţia este generată utilizând forţa de atracţie dintre electromagneţii statorului alimentaţi sub curent constant şi rotorul pasiv. In exemplu, rotorul are patru poli iar statorul şase. Cele trei secvenţe: a) b) c) indică cum se orientează rotorul la succesiunea curentului prin bobinele AA', BB' şi CC'. Spre deosebire de motorul pas cu pas, numai o singură bobină din cele trei poate fi alimentată la un moment dat, motiv pentru care cuplul este mai redus iar funcţionarea este imposibilă sub o viteză minimă limită.

3.5.6 Difuzoare electromagnetice şi piezoelectrice

Fig.3-29 prezintă sintetic trei moduri de interfaţare cu microcontrolerul, a celor mai uzuale tipuri de difuzoare (electromagnetic SP1, piezoelectric SG1 şi electronic cu oscilator încorporat SG2). Comanda este identică pentru toate tipurile, difuzorul electromagnetic are impedanţa cuprinsă între 3 şi 750 de ohmi. Valorile mici ale impedanţei necesită rezistenţe de limitare a curentului prin bobină, care de altfel limitează şi volumul semnalului acustic generat. Banda de frecvenţă este largă: 20Hz…20KHz. Buzerele piezoelectrice necesită cameră de rezonanţă şi un circuit care să crească valoarea tensiunii alternative la borne la circa 15…45V. Banda de frecvenţă reprodusă este limitată la 1…5KHz şi depinde mult de caracteristica transformatorului pe miez de ferită care se poate calcula cu ajutorul legii lui Faraday:

$$u = kN (B\ A_c)\ f\ x10^{-8}$$

unde:
u - tensiunea corespunzătoare înfăşurării calculate [V]
k - factor de formă, 4 pentru undă dreptunghiulară, 4.44 pentru undă sinusoidală
B - inducţia magnetică a miezului (maxim 5000 pentru miez de ferită) [gaus]
Ac - aria secţiunii transversale a miezului magnetic [cm^2]
f- frecvenţa de lucru a bobinei sau transformatorului [Hz]
Cunoscând tensiunea la care lucrează înfăşurarea primară a autotransformatorului $U_{26} = 5V – V_{CE\ T2sat}$, tensiunea de saturaţie a tranzistorului de comandă fiind în cazul cel mai bun 0.25V până la 0.4V, (fig.3-29) se poate determina numărul de spire al înfăşurării L_{n26} pentru o inducţie magnetică de lucru de 2500-3000 gauss. Este inducţia tipică pentru oalele de ferită miniatură. Cunoscând tensiunea necesară elementului piezoelectric (cca. 25V pentru un efect acustic suficient de ridicat) se poate determina raportul de transformare:

$$K = V_{36}/V_{26} = 25/5 = 5$$

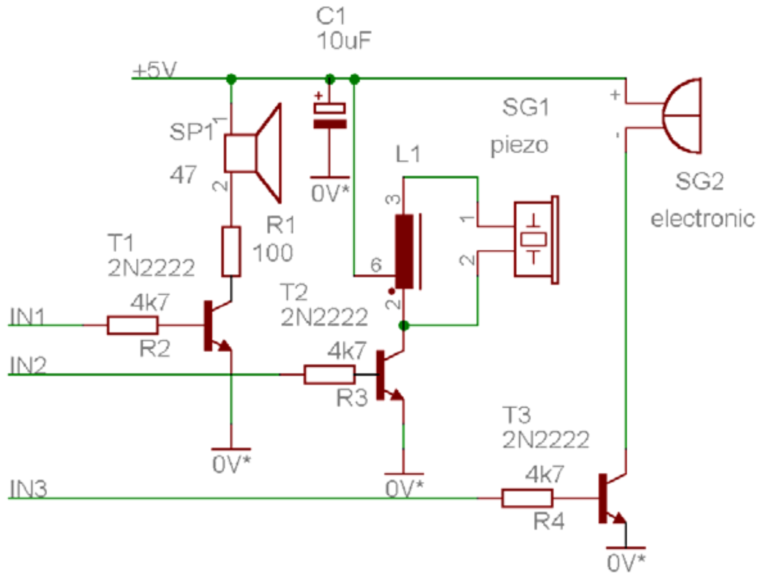


Fig.3-29 Interfaţarea diverselor tipuri de difuzoare

Din acesta rezultă numărul de spire necesar pentru înfăşurarea de tensiune mărită:

$$L_{n36} = k \times L_{n26}$$

Dimensionarea secţiunii conductorului de cupru utilizat pentru bobinare se face cu o relaţie de nomogramă:

$$d = 1.13\sqrt{i/j}$$

unde i – curentul prin înfăşurare [A]
j – densitatea de curent [A/mm^2]

iar din ecuaţia transformatorului: $i_{36} = i_{26}/k$

unde i_{36} – curentul prin înfăşurarea L_{36}
i_{26} – curentul prin înfăşurarea L_{26}

Densitatea de curent la frecvenţe ridicate [kHz] se poate alege la limita superioară a domeniului (1…6A/mm^2) deoarece efectul skin (efectul pelicular, de circulaţie periferică a curentului) este mai pronunţat. Acesta este şi motivul pentru care în cazul curenţilor mari, în locul unui singur fir de cupru se utilizează un mănunchi de fire cu diametre reduse având suma secţiunilor egală sau ceva mai mare decât secţiunea de calcul. Pentru exemplul din figură, secţiunea miezului de ferită necesară alimentării unui element piezoelectric cu diametrul pastilei active de 2cm este sub 0.1 cm^2 (un diametru al feritei de cca. 3mm) iar pentru o dimensionare corectă a numărului de spire se alege frecvenţa minimă de lucru din domeniul preconizat (1KHz…5KHz). Un transformator dimensionat pentru a funcţiona la frecvenţa de 1KHz va funcţiona probabil şi la 5KHz dar cu parametrii de transfer mult coborâţi - tensiune secundară mult redusă.

Buzerele electronice se aseamănă cu cele piezoelectrice cu deosebirea că au oscilatorul pe frecvență fixă în interiorul capsulei rezonante alături de elementul piezoelectric. Polarizarea inversă sau supravoltarea acestor buzere duce la distrugerea lor ireversibilă. O caracteristică importantă a comenzii difuzoarelor este lipsa diodelor supresoare. Utilizarea acestora ar scădea randamentul acustic al difuzorului/buzerului prin limitarea amplitudinii tensiunii. Acest efect se observă cel mai bine la buzerul piezoelectric comandat prin bobină ridicătoare de tensiune. Supresarea oscilațiilor primarului autotransformatorului duce la scăderea randamentului acustic la 50%.

Probabil cititorul se va întreba la ce foloseşte interfațarea unui difuzor direct cu PIC-ul ? Un exemplu este soneria ceasului prezentat în 3.4.2 Dacă se doreşte obținerea unei melodii, se poate folosi modulul CCP1 existent în seria 16F87x sau 16F62x, sunetul reprodus de programul următor nefiind însă polifonic:

```
-- file: music.jal
-- cântă "one litle violin" pe pin_c2 prin PWM hardware
-- compiler: începând de la jal04.24

include f877_04
include jpic
include jdelay

-- fr: frecvența reală
-- fc: frecvența calculată
-- df: eroarea
-- pr2: registrul f877_pr2
-- prs: prescalerul tmr2

-- -----------------------------------------------------------------------
--          OCTAVA1                              OCTAVA2
-- -----------------------------------------------------------------------
--      fr[Hz]  fc[Hz]  Δf[Hz]  pr2/prs  fr[Hz]  fc[Hz]   Δf[Hz]  pr2/prs
--  -----------------------------------------------------------------------
-- Do   261.63  261.5   -0.13   238/16   523.25  525.21  +1.94   118/16
-- Reb  277.18  277.5   +0.32   224/16   554.37  553.09  -1.27   112/16
-- Re   293.66  293.4   -0.26   212/16   587.33  589.62  +2.29   105/16
-- Mib  311.13  310.9   -0.23   200/16   622.25  625     +2.75   99 /16
-- Mi   329.63  328.9   -0.73   189/16   659.26  657.89  -1.36   94 /16
-- Fa   349.23  349.1   -0.13   178/16   698.46  702.24  +3.78   88 /16
-- Fad  369.99  369.8   -0.19   168/16   739.99  744.04  +4.05   83 /16
-- Sol  392     393     +1      158/16   783.99  781.25  -2.74   79 /16
-- Lab  415.3   416.6   +1.3    149/16   830.61  833.33  +2.72   74 /16
-- La   440     440.14  +0.24   141/16   880.0   880.2   +0.2    70 /16
-- Sib  466.16  466.4   +0.24   133/16   932.33  932.8   +0.47   66 /16
-- Si   493.88  492.1   -1.78   126/16   987.77  988     +0.23   254/ 4
-- Do   1046.5  1046.02 -0.47   238/4

procedure frequency ( byte in period, byte in prescale ) is
asm movf  period, w
 bank_1
asm movwf f877_pr2
 bank_0
  f877_ccpr1l = period / 2        -- setează factorul de umplere la aproximativ 0.5
  pin_c2_direction = output
   if prescale == 16 then
```

```
     f877_t2con = 0b_0000_0110    -- tmr2 on, prescale=16
   elsif prescale == 4 then
     f877_t2con = 0b_0000_0101    -- tmr2 on, prescale=4
   elsif prescale == 1 then
     f877_t2con = 0b_0000_0100    -- tmr2 on, prescale=1
   end if
  f877_ccp1con = 0b_0000_1100     -- mod PWM on
end procedure

var byte tempo = 10
var byte measure

procedure l ( byte in time ) is -- lungimea
 measure = tempo / time
 delay_100mS ( measure )
 f877_ccp1con = 0                -- PWM off
end procedure

procedure p ( byte in time ) is -- pauza
 measure = tempo / time
 delay_100mS ( measure )
end procedure

procedure DO1  is frequency ( 238, 16 )  end procedure
procedure REb1 is frequency ( 224, 16 )  end procedure
procedure RE1  is frequency ( 212, 16 )  end procedure
procedure MIb1 is frequency ( 200, 16 )  end procedure
procedure MI1  is frequency ( 189, 16 )  end procedure
procedure FA1  is frequency ( 178, 16 )  end procedure
procedure FAd1 is frequency ( 168, 16 )  end procedure
procedure SOL1 is frequency ( 158, 16 )  end procedure
procedure LAb1 is frequency ( 149, 16 )  end procedure
procedure LA1  is frequency ( 141, 16 )  end procedure
procedure SIb1 is frequency ( 133, 16 )  end procedure
procedure SI1  is frequency ( 126, 16 )  end procedure
procedure DO2  is frequency ( 118, 16 )  end procedure
procedure REb2 is frequency ( 112, 16 )  end procedure
procedure RE2  is frequency ( 105, 16 )  end procedure
procedure MIb2 is frequency (  99, 16 )  end procedure
procedure MI2  is frequency (  94, 16 )  end procedure
procedure FA2  is frequency (  88, 16 )  end procedure
procedure FAd2 is frequency (  83, 16 )  end procedure
procedure SOL2 is frequency (  79, 16 )  end procedure
procedure LAb2 is frequency (  74, 16 )  end procedure
procedure LA2  is frequency (  70, 16 )  end procedure
procedure SIb2 is frequency (  66, 16 )  end procedure
procedure SI2  is frequency ( 252,  4 )  end procedure
procedure DO3  is frequency ( 238,  4 )  end procedure

procedure one_little_violin is
 for 2 loop
  do1 l(8) p(4) re1 l(8) p(4) mi1 l(8) p(4) fa1 l(8) p(4)
  sol1 l(4) p(4) sol1 l(4) p(4) la1 l(4) p(4) la1 l(4) p(4)
  sol1 l(2) p(4)
```

```
 end loop
  fa1 l(4) p(4) fa1 l(4) p(4) mi1 l(4) p(4) mi1 l(4) p(4) re1
l(4) p(4) re1 l(4) p(4) sol1 l(2) p(4)
  fa1 l(4) p(4) fa1 l(4) p(4) mi1 l(4) p(4) mi1 l(4) p(4) re1 l(4)
p(4)re1 l(4) p(4) do1 l(2) p(4)
 for 2 loop
  do2 l(8) p(4) do2 l(8) p(4) si1 l(4) p(4) si1 l(4) p(4) la1 l(4)
p(4)la1 l(4) p(4) sol1 l(2) p(4)
 end loop
  fa1 l(4) p(4) fa1 l(4) p(4) mi1 l(4) p(4) mi1 l(4) p(4) re1 l(4)
p(4) re1 l(4) p(4) sol1 l(2) p(4)
  fa1 l(4) p(4) fa1 l(4) p(4) mi1 l(4) p(4) mi1 l(4) p(4) re1 l(4)
p(4) re1 l(4) p(4) do1 l(2) p(4)
end procedure

-- programul principal începe aici

forever loop
 one_little_violin
                  ; o vioară micăăă de-aş aveaaaa, o vioară micăăăă mi-ar plăceaaaa,
                  ; toată ziua aş cântaaa, multe cântece cu eaaaa,
                  ; trili-li-li-li-li-dum-dum-dum, trili-li-li-li-li-dum-dum-dum….
 delay_1s ( 1 )
end loop
```

In exemplul anterior, **p** reprezintă pauza iar **l** reprezintă durata notei. Argumentul acestor proceduri reprezintă unitatea (1), jumătatea (2), pătrimea (4) sau optimea (8) de întreg şi sunt identice ca valoare atât pentru notă cât şi pentru pauză. O întrebare ar fi: se pot genera sunete polifonice utilizând ambele module CCPx şi mixând semnalul rezultat ? Răspunsul este nu, deoarece frecvenţa celor două PWM–uri este aceeaşi pentru toată seria PIC16, singura soluţie constructivă identică este utilizarea unor microcontrolere din seria PIC18F care au 4 module PWM ce pot avea frecvenţe diferite, sau utilizarea algoritmilor de generare **D**ual **T**one **M**odulated **F**requency (utilizată de telefon) prin software. Să facem împreună o trecere succintă în revistă a modulului CCP1 pentru funcţia de generare PWM: modulul produce un semnal PWM cu o rezoluţie de maxim 10 biţi. Sunt disponibile 1024 de stări distincte pentru factorul de umplere la o frecvenţă dată (fig.3-30).

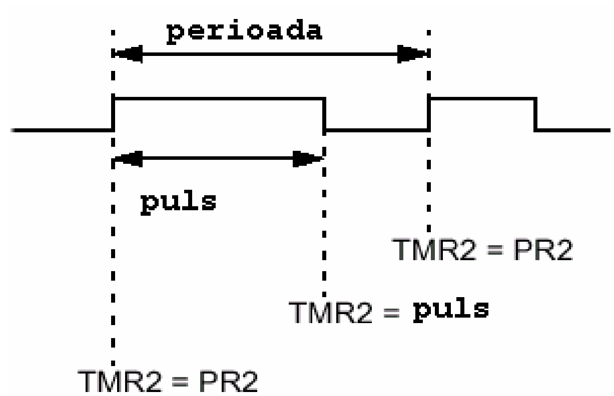


Fig.3-30 Definirea semnalului PWM generat hardware

Sunt două ecuații importante de cunoscut pentru utilizatorul acestui modul, prima este:

Perioada_PWM = [PR2 +1] * 4 Tosc * TMR2_prescaler_value respectiv:
f = 1/perioada_PWM

Din expresiile de mai sus rezultă:
PR2 = [1/ (f * 4 Tosc * TMR2_prescaler_value)] – 1
Unde : Tosc [nS];
f[Hz];
PR2[adimensional];
TMR2_prescaler _value = 16 sau 4 sau 1

Pentru cazul de față Tosc = 1/4MHz = 250 nS (4* Tosc = 1μS)
Cea de-a doua relație importantă este expresia factorului de umplere:
PWM_duty_cycle = [CCPR1L:CCP1CON<5,4>] * Tosc * TMR2_prescaler_value

-	-	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0
7	6	5 R/W	4 R/W	3 R/W	2 R/W	1 R/W	0 R/W
CCPxX:CCPxY: cei mai puțin semnificativi biți ai PWM Cei mai semnificativi biți ai PWM se găsesc în CCPRxL							
CCPxM3:CCPxM0: biții de selecție ai CCPx 0000 = modul captură/comparare/PWM este inactiv 11xx = mod PWM activ CCP1CON							

Fig.3-31 Registrul CCP1CON pentru modul PWM

După cum se poate observa, regiştrii asociați celor două module CCP (pentru PIC16F87x, respectiv al unui singur modul pentru PIC16F62x) sunt: CCPR1L, CCPR1H, CCP1CON respectiv CCPR2L, CCPR2H şi CCP2CON.

CCPRxL conține cei mai semnificativi 8 biți ai factorului de umplere, în timp ce biții 5 şi 4 ai CCPxCON conțin cei mai puțin semnificativi 2 biți. Registrul CCPxH este folosit împreună cu un *latch* intern de 2 biți pentru a copia valoarea registrului CCPxL, funcție necesară pentru a elimina *glitch*-urile de ieşire ale semnalului PWM. Din această cauză în modul PWM, registrul CCPxH nu poate fi scris ci doar citit. Deoarece timerul asociat modului PWM este timerul2, este necesară setarea regiştrilor corespunzători acestui timer şi anume T2CON, TMR2 şi PR2. După cum ați văzut în programul exemplificat, perioada PWM trebuie înscrisă în registrul PR2. Când valoarea din TMR2 este egală cu valoarea înscrisă în registrul PR2 se generează trei evenimente:

- ♦ TMR2 este resetat
- ♦ Pinul CCP1 (pe care iese semnalul PWM) devine high (mai puțin când factorul de umplere sau *duty_cycle* = 0)
- ♦ Valoarea factorului de umplere (*duty_cycle)* este transferată din CCPxL în CCPxH pentru a preîntîmpina apariția *glitch*-urilor.

Comparatorul intern compară valoarea CCPxH cu valoarea TMR2 şi activează sau nu bistabilul de ieşire ce comandă pinul ieşirii PWM. Bineînțeles că direcția pinului a fost setată în prealabil ca ieşire din registrul TRISx asociat.

O obsevaţie cauzatoare de neplăceri dacă nu este luată în calcul, este faptul că nu se pot obţine semnale PWM cu rezoluţia de 10 biţi pentru orice frecvenţe ale semnalului. De exemplu un semnal PWM cu frecvenţa de 200KHz, obţinut dintr-un PIC16F877 rulând la 20MHz, va avea rezoluţia de doar 5…6 biţi. Expresia ce calculează numărul de biţi raportat la frecvenţa PWM este:

$$rezolutia = \frac{\log(\frac{Fosc}{Fpwm})}{\log(2)} biti$$

Este important ca factorul de umplere al PWM să nu fie setat mai lung decât perioada semnalului PWM deoarece, pentru această situaţie, registrul CCPx nu va fi şters şi ca urmare semnalul PWM nu va fi generat.

Transferul programului demonstrativ prezentat, de pe PIC16F87x pe PIC16F628 este simplu: se va avea în vedere schimbarea denumirii pinului PWM corespunzător şi modificarea bibliotecilor incluse (fila de definire a procesorului şi biblioteca jpic), cât şi dezactivarea comparatoarelor interne ale PIC16F628 (dacă acestea nu se utilizează), prin setarea convenabilă a registrului CMCON (CMCON = 7).

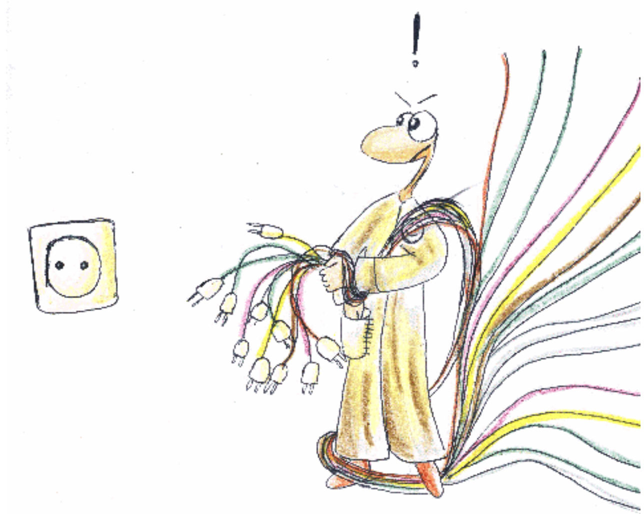
-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
7	6 R/W	5 R/W	4 R/W	3 R/W	2 R/W	1 R/W	0 R/W
TOUTPS3:TOUTPS0: biţi de selecţie ai postscaler-ului TMR2 0000 = 1:1 0001 = 1:2 … 1111 = 1:16							
TMR2ON: bitul de startare al TMR2 1 = TMR2 este pornit 0 = TMR2 este oprit							
T2CKPS1:T2CKPS0: prescaler pentru TMR2 00 = prescaler 1 01 = prescaler 4 1x = prescaler 16							T2CON

Fig.3-32 Registrul T2CON

Bibliografie:

1. Cifru electronic cu un singur buton: http://www.voti.nl
2. Mike Predko, PICmicro MCU, Application Design and Hardware Interfacing, http://www.mike.com
3. MMC22925/MMC22926 filă de catalog, CMOS Data Book, Microelectronica Bucureşti, 1992
4. PIC16F87x datasheet, DS30292C, Microchip Technology Inc. 2001
5. Electronic Design, september, 1980, The switcher transformer: Designing it in one try for switching power supplies

4 Interfaţarea circuitelor integrate "inteligente"

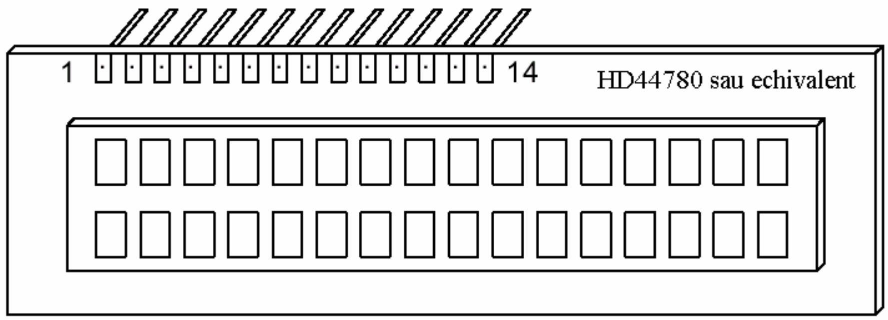


Se zvoneşte că omul este singura fiinţă ceva mai inteligentă ce populează planeta. Discutabil. Un **circuit integrat inteligent** este rodul creaţiei acestui personaj controversat. Se recunoaşte foarte uşor după numărul impresionant de pagini conţinut în fila de catalog şi notele de aplicaţie. Primul contact al utilizatorului cu informaţia referitoare la acest tip de integrat îi creează un puternic sentiment de frustrare şi deznădejde. Cel ce renunţă uşor nu va trece niciodată mai departe de noţiunile introductive. Un utilizator exuberant va începe însă cu notele de aplicaţie. Inţelegerea deplină şi totală a funcţionării acestor integrate are loc doar după ce un produs de serie părăseşte atelierul creatorului şi acesta se confruntă în mod direct şi personal cu toate problemele mărunte ale existenţei sale de utilizator al circuitului integrat inteligent. Dezamăgiţi ? Sper că nu încă…Din această categorie (pe care cu greu am definit-o…) se pot enumera: modulele de afişare alfanumerică LCD, afişoare grafice cu LCD, diverse convertoare AD, circuite integrate specializate în măsurarea temperaturii, generatoare de funcţii sau de semnal sinusoidal, memorii EEPROM sau (S)RAM, senzori inteligenţi şi bineînţeles că nu am atins nici 30% din posibilităţi. Este indubitabil că novicele nu va deveni specialist peste noapte doar parcurgând aceste rânduri, dar cu siguranţă va şti mai târziu cum să abordeze o problemă cu care se confruntă pentru prima dată în viaţă. Şi o ultimă dorinţă: listarea întregului document în format PDF corespunzător circuitului în discuţie, de pe CD-ul anexat sau web, este absolut obligatorie !

4.1 Afişaj inteligent alfanumeric cu cristale lichide, compatibil HD44780

Driverul Hitachi HD44780 este unul dintre cele mai cunoscute circuite integrate existente la ora actuală pe glob, destinat modulelor de afişare cu simboluri alfanumerice formate din matrice de puncte (dot matrix). Aspectul tipic al unui afişaj de tip *dot matrix* cu două rânduri şi 16 caractere pe rând este cel din fig.4-1. Pot exista diferenţe de amplasare a conectorului cu 14 pini, acesta poate fi amplasat şi în stânga sau dreapta jos pe direcţia normală de vizare sau pe două rânduri de 8 pini pe laterala N-S a afişajului. După vizibilitatea caracterelor se deosebesc afişaje cu vizibilitate standard de la ora 12 sau ora 6 în funcţie de unghiul de vizare (tipic 30…40^{o}) raportat la axa ochilor, respectiv afişaje *supertwist* cu unghi de vizare dublu a cărui aspect rămâne neschimbat dacă se priveşte atât dinspre ora 6 cât şi dinspre ora 12. După modul de formare a imaginii există module transreflective care nu necesită iluminare din spate şi reflective cu *backlight* care dispun de o sursă proprie de iluminare cu LED-uri sau folie electroluminiscentă. După culoarea

imaginii observată de ochiul subiectului, sunt afişoare normale (matrice de puncte întunecate pe fond luminos) sau inverse (matrice de puncte luminoase pe fond întunecat, numai la module reflective). Modulele cu *backlight* au doi conectori suplimentari care pot apărea în continuarea celor 14 sau separat pe marginea N-S a modulului, aceştia sunt utilizați pentru alimentarea LED-urilor sau a foliei electroluminiscente.



PIN	SIMBOL	FUNCTIA
1	Vss	masa
2	Vdd	+5V
3	Vo	ajustare contrast Vlc
4	RS	0 = instructiune, 1= data
5	R / W	0 = scrie, 1= citeste
6	E	afisare permisa (enable)
7	D0	Data 0
8	D1	Data 1
9	D2	Data 2
10	D3	Data 3
11	D4	Data 4
12	D5	Data 5
13	D6	Data 6
14	D7	Data 7

fig.4-1 Conexiunile modulului LCD cu aranjament “inline”

4.1.1 Regiştrii HD44780

HD44780 au doi regiştrii de 8 biți [1]: un registru de instrucțiune (***I****nstruction* ***R****egister*) şi un registru de date (***D****ata* ***R****egister*). Registrul IR memorează codul instrucțiunii: ştergerea afişajului sau rotirea cursorului, informația de adresă pentru afişarea datelor în RAM (**D**ata **D**isplay RAM) sau afişarea datelor în generatorul de caractere (**C**haracter **G**enerator RAM). Registrul IR poate fi doar scris de către microcontroler. Registrul de date DR memorează temporar datele ce urmează să fie scrise sau citite în/din DDRAM sau CGRAM, de operațiunea internă realizată de driverul HD44780. Când adresa este scrisă în registrul IR, data este citită automat în DR din DD RAM sau CG RAM prin operațiunea internă de care aminteam. Transferul de date spre microcontroler este terminat de acesta prin citirea registrului DR. După citire, data provenită din DD RAM sau CG RAM, corespunzătoare următoarei adrese este trimisă în registrul DR. Semnalul de selecție al registrului (***R****egister* ***S****elector*) face deosebirea între cei doi regiştrii:

```
RS  R/W  Enable   Operaţia
==  ===  ======   =========
 0   0   H,H->L  IR scrie ca operaţiune internă  (display, clear, etc.)
 0   1     H     Citeşte busy flag (DB7) şi numărătorul de adresă (DB0-DB6)
 1   0   H,H->L  DR scrie ca operaţiune internă (DR spre DD RAM sau CG RAM)
 1   1     H     DR citeşte ca operaţiune internă  (DD RAM sau CG RAM spre DR)
```

Busy Flag

Când flagul *busy* este *high,* HD44780 este ocupat cu modul de operare intern şi următoarea instrucţiune de la microcontroler nu va fi acceptată. *Busy flag* este direcţionat spre ieşirea DB7 când RS = 0 şi R/W = 1. Următoarea instrucţiune trebuie scrisă numai după ce s-a verificat că *busy flag* este *low.*

Numărătorul de adrese (Address Counter)

Numărătorul de adrese AC asignează adrese fie pentru memoria de date DD RAM, fie pentru memoria de caractere CG RAM. Când o instrucţiune de adresare este scrisă în registrul de instrucţiune IR, informaţia este trimisă din IR în numărătorul de adrese AC. Selecţia memoriei DD RAM sau CG RAM este deasemenea determinată preferenţial de instrucţiune. După o scriere sau o citire în/din DD RAM sau CG RAM, numărătorul de adrese AC este incrementat sau decrementat automat cu 1. Conţinutul numărătorului de adrese este accesibil pe liniile DB0-DB6 când RS = 0 şi R/W = 1 ca în tabelul de mai sus.

Memoria de date RAM (DD RAM)

Memoria DD RAM memorează datele ce urmează să fie afişate, reprezentate în coduri ale caracterului pe 8 biţi. Capacitatea sa este de 80 x 8 biţi sau 80 de caractere. Pe un afişaj cu mai puţin de 80 de caractere, orice locaţie DD RAM care nu este utilizată poate fi folosită ca memorie RAM de uz general. Relaţia tipică (dar nu comună tuturor modulelor LCD) dintre adresa memoriei DD RAM şi poziţia caracterului pe afişajul cu cristal lichid HD44780, 2x16, este prezentată în tabelul următor. (Adresa DD RAM este setată în numărătorul de adrese AC în format hexazecimal)

Poziţia	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Linia1	80	81	82	83	84	85	86	87
Linia2	C0	C1	C2	C3	C4	C5	C6	C7
Poziţia	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**
Linia1	88	89	8A	8B	8C	8D	8E	8F
Linia2	C8	C9	CA	CB	CC	CD	CE	CF

Generatorul de caractere ROM (Character Generator ROM)

Generatorul de caractere poate genera matrici de 5 x 7 puncte sau 5 x 10 puncte din codul caracterului de 8 biţi. Conţine de regulă 192 de caractere 5 x 7 şi 192 de caractere 5 x 10.

Generatorul de caractere RAM (Character Generator RAM)

Generatorul de caractere RAM este o porţiune din memoria RAM unde utilizatorul poate redefini prin software aspectul caracterului. Pentru matricea de 5 x 7 puncte se pot defini 8 caractere utilizator iar pentru matricea 5 x 10 puncte doar 4.

4.1.2 Setul de instrucțiuni HD44780

Setul de instrucțiuni al HD44780 şi compatibile (KS0066, T7934, 6426), timpul de execuție se referă la modul 8 biți respectiv 4 biți de date.

Instrucțiune	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
Clear Display	0	0	0	0	0	0	0	0	0	1
Return Home	0	0	0	0	0	0	0	0	1	*
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	S
Display ON/OFF	0	0	0	0	0	0	1	D	C	B
Cursor and Display Shift	0	0	0	0	0	1	S/C	R/L	*	*
Function Set	0	0	0	0	1	DL	N	F	*	*
Set CG RAM address	0	0	0	1	A	A	A	A	A	A
Set DD RAM address	0	0	1	A	A	A	A	A	A	A
Read busy flag and address	0	1	BF	A	A	A	A	A	A	A
Write data to CG or DD RAM	1	0	D	D	D	D	D	D	D	D
Read data from CG or DD RAM	1	1	D	D	D	D	D	D	D	D

Clear Display

Sterge întregul afişaj. Scrie codul 20h (*blank*) în toate locațiile DD RAM. Setează adresa DD RAM la 0 în numărătorul de adrese. Se revine cu afişajul în poziția inițială, adică afişarea dispare şi cursorul se deplasează în prima linie la prima poziție. Setează I/D = 1 (***I**ncrement **M**ode*). Timp de execuție = 82µs-1.64ms / 120µs-4.9ms

Return Home

Pune cursorul în poziția *home* (adresa 0) şi afişajul dacă a fost deplasat este returnat în poziția inițială. Conținutul DD RAM rămâne neschimbat. Timp de execuție: 40µs-1.6ms / 120µs-4.8ms

Entry Mode Set

Setează direcția de mişcare a cursorului şi specifică sau nu rotirea conținutului afişajului. Aceste operații sunt efectuate pe timpul scrierii şi citirii datelor. I/D: **I**ncrementează (I/D = 1) sau **D**ecrementează (I/D = 0) adresa DD RAM cu 1 când codul unui caracter este scris sau citit. Cursorul sau pâlpâirea caracterului se mută spre dreapta când este incrementat cu 1 sau spre stânga când este decrementat cu 1. Aceleaşi modificări se aplică citirii sau scrierii din CG RAM. ***S**hift*; când S = 1 roteşte întregul afişaj fie spre stânga (I/D = 1), fie spre dreapta (I/D = 0), nu roteşte afişajul pentru S = 0. Aspectul vizibil este ca şi când cusorul stă pe loc şi mesajul se mişcă. Timp de execuție: 40µs / 120µs

Display ON/OFF

Setează afişajul ON/OFF, cursorul şi poziția cursorului care pâlpâie.
D: Afişajul este ON când D = 1 şi OFF când D = 0. După o comandă D = 0, datele afişate rămân în DD RAM şi pot fi afişate imediat setând D = 1.
C: Cursorul este afişat pentru C = 1 şi nu este vizibil pentru C = 0. Chiar dacă cursorul dispare, funcția I/D nu se schimbă pe parcursul scrierii datelor pe afişaj. Cursorul este afişat

folosind 5 puncte în cea de-a opta linie pentru caracterul format din 5x7 puncte sau din 5 puncte în cea de-a 11 linie pentru caracterul de 5x10 puncte.
B: Caracterul indicat de cursor pâlpâie când B = 1 şi este afişat normal pentru B = 0. Pâlpâirea este afişată prin schimbarea culorii tuturor punctelor albe şi afişarea/stingerea caracterului la un interval de cca 400mS.
Timp de execuţie: 40µs / 120µs

Cursor and Display Shift

Mută cursorul şi roteşte afişajul fără a schimba conţinutul DD RAM. Mutarea cursorului se face fără citirea sau scrierea de date. Această funcţie este utilizată pentru a corecta sau a căuta un caracter afişat. Cursorul se mută în cea de-a doua linie când trece de ultimul digit al primei linii (linie de 8, 16, 20 sau 40 caractere). De observat că prima şi a doua linie afişată se va roti în acelaşi timp. Când data afişată este rotită în mod repetat, fiecare linie se mişcă doar orizontal. A doua linie nu poate să se rotească în poziţia primei linii.

```
S/C R/L
=== ===
 0   0  Roteşte poziţia cursorului spre stânga
        (Address Counter este decrementat cu 1)
 0   1  Roteşte poziţia cursorului spre dreapta
        (Address Counter este incrementat cu 1)
 1   0  Roteşte întregul afişaj spre stânga
        Cursorul urmăreşte rotaţia afişajului
 1   1  Roteşte întregul afişaj spre dreapta
        Cursorul urmăreşte rotaţia afişajului
```

Timp de execuţie: 40µs / 120µs

Function Set

Setează lungimea datelor (DL), numărul de linii afişat (N) şi fontul caracterelor (F)
DL setează lungimea datelor :

- Data este transmisă sau recepţionată în modul 8 biţi (DB7-DB0) când DL = 1
- Data este transmisă sau recepţionată în modul 4 biţi (DB7-DB4) când DL = 0
- Când se lucrează în modul 4biţi, data se transmite sau se recepţionează de două ori: lsn şi msn *(**l**ast **s**emnificative **n**ibble, **m**ost **s**emnificative **n**ibble*)

N: Setează numărul de linii al afişajului, N = 1 două linii, N = 0 o linie
F: Setează fontul caracterelor, F = 1 caracter de 5x10 pixeli, F = 0 caracter de 5x7 pixeli
Instrucţiunea *funcţion set* trebuie executată la începutul programului înaintea orcărei alte instrucţiuni (cu excepţia *read busy flag and address)*. După momentul primei execuţii, instrucţiunea *function set* nu poate fi executată din nou până când este modificată lungimea de comunicaţie a datelor din bitul DL (4 sau 8 biţi).

N F	afişare linii	fontul caracter	factor umplere	obs.
0 0	1	5x 7 dots	1/8	-
0 1	1	5x10 dots	1/11	-
1 *	2	5x 7 dots	1/16	nu poate afişa 2 linii 5x10

Timp de execuţie 40µs / 120µs

Set CG RAM address

Setează adresa CG RAM în numărătorul de adrese în forma binară 0001AAAAAA. Data este apoi scrisă sau citită dinspre microcontroler pentru CG RAM. Timp de execuție: 40µs / 120µs

Set DD RAM address

Setează adresa DD RAM în numărătorul de adrese în format binar: 001AAAAAAA. Data este apoi scrisă sau citită de la microcontroler pentru DD RAM.

Când N=0 (afişajul are 1 linie) adresa poate fi în domeniul: 00h-4Fh, 80h-C7h pentru 16 caractere/rând
Când N=1(afişajul are 2 linii) adresa poate fi pentru prima linie: 00h–27h, 80h-8Fh (16caractere/rând), 80h-93h (20 caractere/rând), 80h-A7h (40 caractere/rând), respectiv pentru cea de-a doua linie: 40h–67h, C0h-CFh (16 caractere/rând), C0h-D3h, A0h-B3h (20 caractere/rând), C0h-E7h (40 caractere/rând).
Timp de execuție: 40µs / 120µs

Read busy flag and address

Citirea BF indică faptul că sistemul execută intern o instrucțiune anterioară. BF=1 indică o operație internă în desfăşurare. Următoarea instrucțiune nu va fi acceptată până când BF=0. Verificați că BF=0 înaintea următoarei operații de scriere. In acelaşi moment cu verificarea, este citită valoarea numărătorului de adresă din 01(BF)AAAAAAA. Numărătorul de adresă este utilizat de ambele adrese CG RAM şi DD RAM. Folosirea lui curentă este determinată de instrucțiunea anterioară. Timp de execuție: 1µs

Write data to CG or DD RAM

Scrie 8 biți de date DDDDDDDD în memoria CG RAM sau DD RAM. Dacă se va scrie în CG RAM sau DD RAM este determinat de specificația anterioară a adresei CG RAM sau DD RAM. După scriere, adresa este incrementată sau decrementată automat cu 1 în funcție de *entry mode set* care determină şi rotirea conținutului afişajului. Timp de execuție: 40µs / 120µs

Read data from CG or DD RAM

Citeşte valoarea octetului DDDDDDDD din memoria CG RAM sau DD RAM. Adresarea anterioară spune dacă va fi citită memoria CG RAM sau DD RAM. Inaintea introducerii instrucțiunii de citire trebuie executată instrucțiunea de setare a adresei CG RAM sau DD RAM. Dacă nu este setată nici o adresă, prima dată citită nu va fi validă. Când se execută în mod repetat instrucțiunea de citire, data este citită la pasul următor.
Dacă rotirea cursorului se face cu instrucțiunea *cursor shift* şi se citeşte conținutul DD RAM, nu e nevoie să fie executată instrucțiunea *set CG RAM / DD RAM address* imediat înaintea instrucțiunii de citire. Instrucțiunea *cursor shift* are acelaşi efect ca instrucțiunea de setare a adresei DD RAM. După o citire, instrucțiunea *entry mode* incrementează sau decrementează automat adresa cu 1. De reținut că rotirea conținutului afişajului nu se execută în acest moment indiferent ce mod este setat.
Numărătorul de adresă AC este incrementat/decrementat automat (în funcție de *entry mode set*) după o instrucțiune de scriere fie în CG RAM fie în DD RAM. Data din memoria RAM selectată de numărătorul de adresă AC, nu poate fi citită chiar dacă se execută imediat o

operaţie de citire. Condiţia pentru o citire corectă a datei este executarea instrucţiunii de setare a adresei sau de rotire a cursorului (numai cu DD RAM) chiar înainte de execuţia instrucţiunii de citire. Timp de execuţie: 40µs / 120µs.

4.1.3 Iniţializarea HD44780

Iniţializarea prin resetare internă

HD44780 se iniţializează automat la alimentarea cipului. BF este menţinut în starea busy până la terminarea iniţializării. Acestă stare durează 10mS până când tensiunea de alimentare Vcc creşte peste 4,5V. Următoarele instrucţiuni sunt executate în faza de iniţializare:

```
1. Display clear
2. Function set ..... DL = 1: interfaţa de 8 biţi
                         N = 0: afişaj cu o linie
                         F = 1: font 5 x 10 pixeli

3. Display ON/OFF ...  D = 0: display OFF
                       C = 0: cursor OFF
                       B = 0: blink OFF

4. Entry mode set .. I/D = 1: +1 (increment)
                       S = 0: fără rotire
5. Write DD RAM
```

Când timpul de creştere a tensiunii de alimentare depăşeşte 10mS sau tensiunea pe afişaj este mai mare de 0.2V în stare nealimentată, circuitul de reset nu va funcţiona corect şi iniţializarea nu va avea loc în mod corespunzător. In această situaţie se impune efectuarea iniţializării software.

Observaţie: unele afişaje pot avea inţializarea hardware uşor diferită (numărul de linii sau fontul caracterului)

Iniţializarea software pentru o interfaţă de 8 biţi:

[alimentare ON]
[aşteptare mai mult de 15ms după ce VDD > 4.5V]

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	BF nu se poate verifica înainte
0	0	0	0	1	1	*	*	*	*	*Function set* pentru interfaţă de 8-biţi

[aşteaptă mai mult de 4.1ms]

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	BF nu se poate verifica înainte
0	0	0	0	1	1	*	*	*	*	*Function set* pentru interfaţă de 8-biţi

[aşteaptă mai mult de100us]

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	BF nu se poate verifica înainte
0	0	0	0	1	1	*	*	*	*	*Function set* pentru interfaţă de 8-biţi

BF poate fi verificat după următoarele instrucţiuni. Când BF nu este verificat, timpul de aşteptare dintre instrucţiuni este mai lung decât timpul de execuţie al instrucţiunii.

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
0	0	0	0	1	1	N	F	*	*	*Function set* pentru interfaţă de 8-biţi Specifică nr. de linii şi fontul caracterului, ultima modificare posibilă

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
0	0	0	0	0	0	1	0	0	0	Afişare OFF , cursor OFF, blink OFF

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
0	0	0	0	0	0	1	1	0	0	Afişare ON, cursor OFF, blink OFF

RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
0	0	0	0	0	0	0	1	I/D	S	setare *entry mode*

[sfârşitul iniţializării 8 biţi]

Iniţializarea software pentru o interfaţă de 4 biţi:

[alimentare ON]
[aşteaptă mai mult de 15ms după ce Vdd > 4.5v]

RS	R/W	DB7	DB6	DB5	DB4	BF nu se poate verifica înainte
0	0	0	0	1	1	*Function set* pentru interfaţă de 8-biţi

[aşteaptă mai mult de 4.1ms]

RS	R/W	DB7	DB6	DB5	DB4	BF nu se poate verifica înainte
0	0	0	0	1	1	*Function set* pentru interfaţă de 8-biţi

[aşteaptă mai mult de 100us]

RS	R/W	DB7	DB6	DB5	DB4	BF nu se poate verifica înainte
0	0	0	0	1	1	*Function set* pentru interfaţă de 8-biţi

BF poate fi verificat după aceste instrucţiuni. Când BF nu este verificat, timpul de aşteptare dintre instrucţiuni este mai lung decât timpul de execuţie al instrucţiunilor.

RS	R/W	DB7	DB6	DB5	DB4	
0	0	0	0	1	0	*Function set* pentru interfaţă de 4-biţi

RS	R/W	DB7	DB6	DB5	DB4	
0	0	0	0	1	0	
0	0	N	F	*	*	*Function set* pentru interfaţă de 4-biţi, specifică nr. de linii şi fontul caracterelor; nu mai pot fi modificate de aici înainte

RS	R/W	DB7	DB6	DB5	DB4	
0	0	0	0	0	0	
0	0	1	0	0	0	Afişaj OFF , cursor OFF, blink OFF

RS	R/W	DB7	DB6	DB5	DB4	
0	0	0	0	0	0	
0	0	1	1	0	0	Afişaj ON, cursor OFF, blink OFF

RS	R/W	DB7	DB6	DB5	DB4

```
0    0    0    0    0    0
0    0    0    1   I/D   S
```
setează *entry mode*

[sfârşitul inițializării 4 biți]

Cititorul dispune de biblioteca hd447804.jal respectiv hd447808.jal care realizează ambele inițializări pentru modul de conexiune, numai pentru scriere în LCD. Dacă doriți să lucrați în modul testare *busy-flag,* atunci biblioteca trebuie modificată.

Observație: Informațiile prezentate mai sus sunt aplicabile tuturor afişajelor Seiko, compatibile HD44780. Acestea sunt: M1641, L1651, M1632, L1642, L1652, L2012, L2432, L4042, L1614, L2014, M4024.
Modulele LCD cu care am lucrat şi care **sunt total compatibile** cu algoritmul de mai sus, au fost: HD44780 2x16, KS0066 (modul KP-01) 2x16,1x16, T7934 1x16, şi M6426 (modul NEC02070AA) 4x20. Pentru acestea, denumirea este de fapt numele driverului existent pe placa PCB a afişajului. Diferențele între module sunt reprezentate doar de potențialul necesar pentru contrast (VLC fig.4-2 sau fig.4-1) şi adresa caracterelor în CG RAM.

4.2 Interfațarea LCD-ului inteligent în modul 6 fire (4date + 2 comenzi)

Modul de interfațare economic se utilizează acolo unde nu este nevoie de o magistrală de date de 8 biți, a cărui funcționalitate să fie împărțită între dispozitivul de afişare şi altă componentă similară, (cum ar fi memoria SRAM cu acces paralel) iar microcontrolerul utilizat are un număr redus de pini. Şi magistrala de 4 biți poate fi utilizată de mai multe dispozitive inteligente sau combinații între acestea şi butoane independente (vezi cap.4.2.3) sau keypad-uri. Dacă nu intenționăm să citim informația provenită de la LCD, ci doar să scriem date spre el, atunci mai e nevoie de încă două linii de comandă: **EN**able şi **R**egister **S**elect, pinul **W**rite-**R**ead fiind conectat la masă (fig.4-2). Este importantă asigurarea tensiunii de contrast pe pinul **VLC,** printr-un potențiometru semireglabil divizor. Majoritatea afişajelor LCD necesită un potențial de cca. +1.8…+2.5V pe acest pin. Fără a iniția comunicația cu microcontrolerul, se alimentează afişajul LCD şi se reglează din R2 până când se observă rețeaua de puncte ce formează matricea caracterului afişat. De obicei sunt vizibile matricile corespunzătoare primului rând al afişajului (primele 8 caractere pentru afişajul cu 1 x 16 caractere, primul rând pentru afişajul cu 2 x 16 caractere sau 4 x 20 caractere etc). Dacă totul este corect şi nu observăm nimic pe afişaj, putem avea de a face cu un afişaj care necesită tensiune negativă pe pinul VLC (-3V…-5V). Biblioteca ce conține definirea pinilor microcontrolerului implicați în proiect trebuie scrisă sau modificată corespunzător. Inițial aceasta se numeşte HD44780p.jal iar pentru schema din fig.4-2 ea devine LCD6Wp.jal.

```
-- fila        : LCD6Wp.jal
-- data        : 21-may-2001
-- folosit de: hd447804.jal
-- IMPORTANT  : include hd44780p trebuie marcată ca şi comentariu în hd447804.lib
```

```
-- pini       :
-- PIC16F84    HD44780
-- -------------------------
--  5 Gnd       1 Gnd
-- 14 Vcc       2 Vcc
--  3 Contrast
--  6 B0       11 D4
--  7 B1       12 D5
--  8 B2       13 D6
--  9 B3       14 D7
-- 10 B4        4 RS D/I
-- 13 B7        6 EN

var volatile bit  hd44780_4_DI is pin_b4
var volatile bit  hd44780_4_E  is pin_b7
var volatile byte hd44780_4_D  is port_b_low

 procedure _hd44780_4_init is
   port_b_low              = 0
   pin_b4                  = low
   pin_b7                  = low
   port_b_low_direction    = all_output
   pin_b4_direction        = output
   pin_b7_direction        = output
 end procedure
```

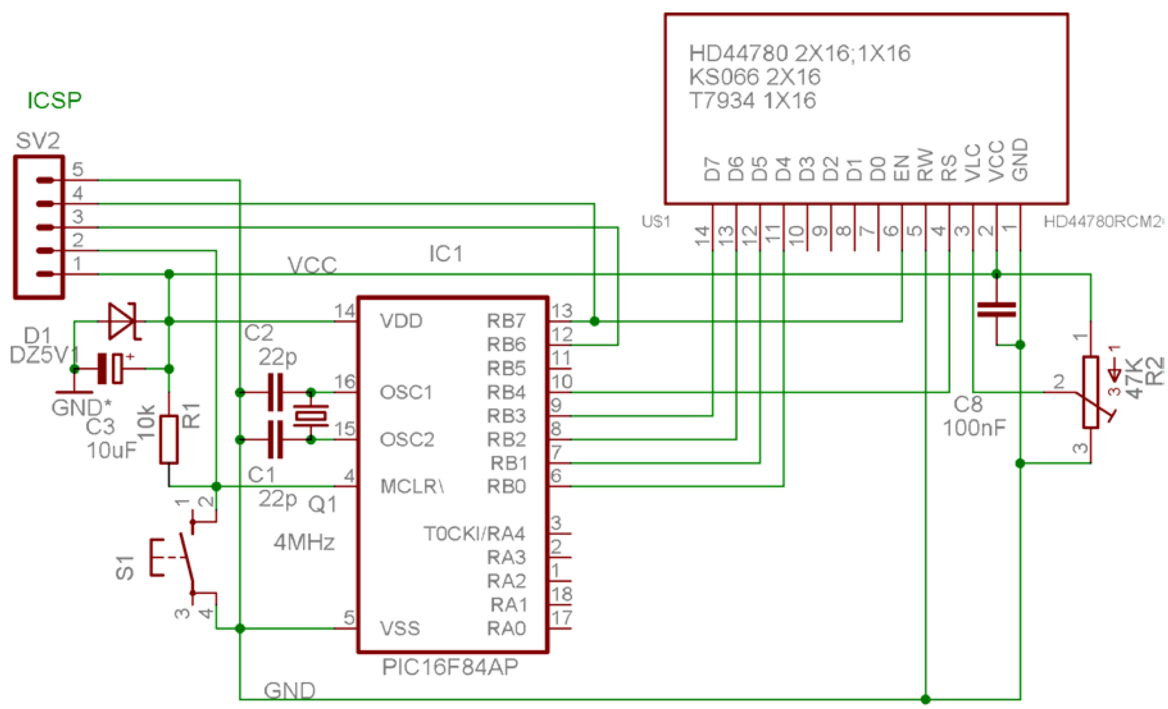


fig.4- 2 Modul de interfațare 4 + 2

Pentru programul de test al corectitudinii conexiunilor se poate utiliza unul foarte simplu:

```
include jpic
include LCD6Wp
include hd447804
HD44780_clear ; inițializarea modulului
HD44780_line1 ; cursor linia1 poziția 0

HD44780=“C” HD44780=“e” HD44780=“” HD44780=“F” HD44780=“a”
HD44780=“i” HD44780=“n” HD44780=“” HD44780=“!”
```

Dacă se doreşte identificarea setului de caractere al afişajului disponibil (CG RAM), aceasta se face printr-un programel foarte scurt:

```
include jpic
include LCD6Wp
include hd447804
include jprint
include jdelay
var byte a = 0
HD44780_clear

for 255 loop
  HD44780_line1 ; pe linia întâi poziția caracterului
  print_hexadecimal2 ( HD44780, a, “0” )
  HD44780_line2 ; pe linia a doua cum arată caracterul
  HD44780 = a
  delay_100mS ( 3 ); delay necesar să vedem ceva pe afişaj…
  a = a + 1
end loop
```

Se observă că în exemplul din fig.4-2, **EN**able este conectat pe unul din pinii de programare ai microcontrolerului. Dacă afişajul LCD este alimentat din aceeaşi linie de alimentare cu +5V ca şi microcontrolerul, programatorul de microcontrolere trebuie să asigure întregul curent de alimentare destinat programării, cât şi cel necesar alimentării afişajului. Dacă dispuneți de un afişaj energofag (curent de alimentare mare), înscrierea microcontrolerului prin ICSP nu este posibilă. Există cel puțin două soluții elegante de remediere a problemei:

- Separarea alimentării microcontrolerului de cea a afişajului printr-o diodă, astfel încât programatorul să asigure tensiune de alimentare doar microcontrolerului. Deoarece microcontrolerul funcționează şi la tensiuni de alimentare de 4.3V, căderea de tensiune pe diodă nu creează probleme.
- Alimentarea circuitului dintr-o sursă proprie care să alimenteze în acelaşi timp şi programatorul paralel (LED-ul verde D2, fig.1-1 va lumina în momentul conectării ICSP în soclul de conexiune spre circuitul de programat).
- Utilizarea unui soclu pentru microcontroler şi programarea lui prin orice altă modalitate decât ICSP

4.3 Interfațarea LCD-ului inteligent în modul 10 fire (8date + 2comenzi)

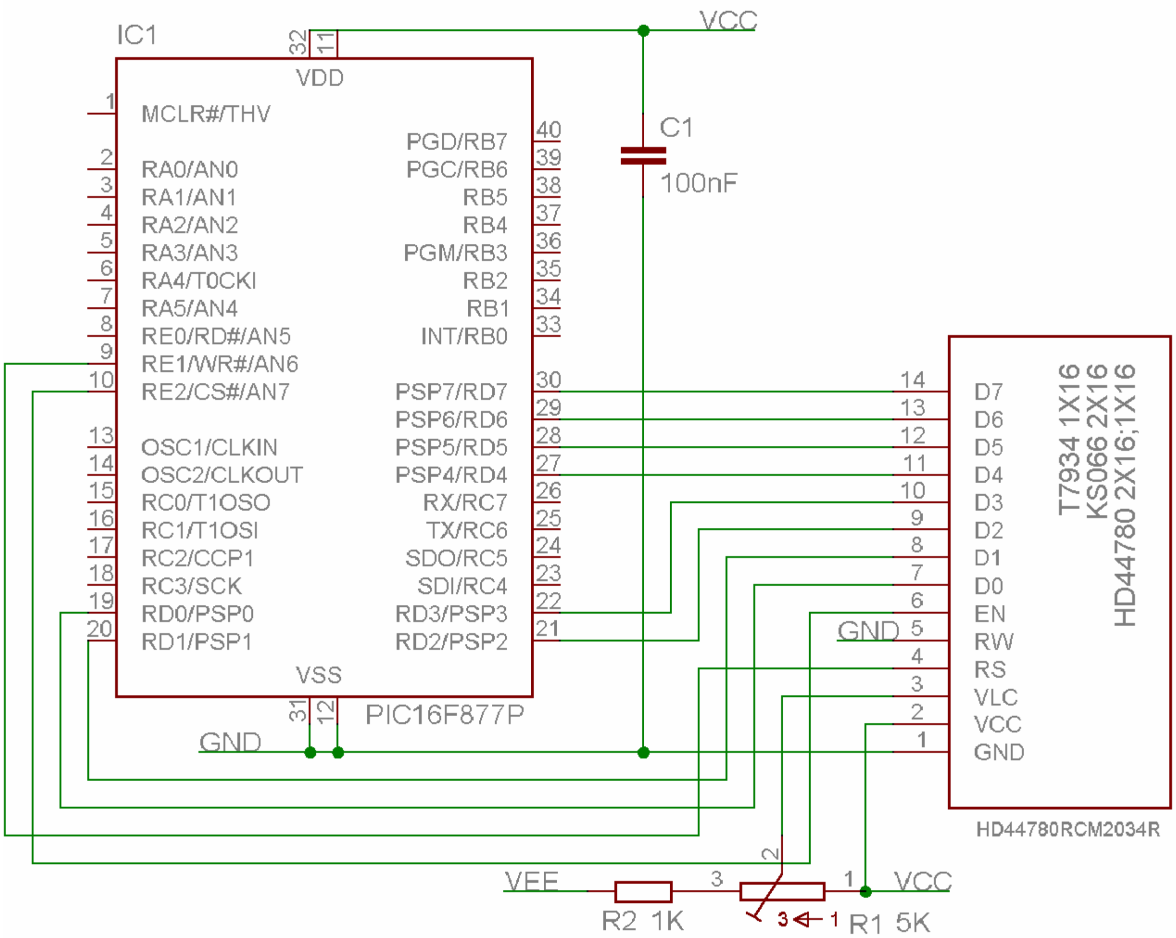


fig.4- 3 Modul de interfațare 8 + 2

Nu există o regulă anume de interfațare a modulului de afișare, însă este util a păstra cei 8 biți de date pe același port, doar pentru simplificarea codului ce trebuie scris. Modul 8 + 2 fire se utilizează de obicei cu microcontrolere având un număr mai mare de pini (28 sau 40/44), însă se poate utiliza cu succes și pentru microcontrolere cu 18 pini, mai ales dacă este necesar un bus paralel de 8 biți. Timpul de procesor necesar scrierii în modulul LCD se micșorează față de modul de interfațare 4 + 2. Afișorul folosit în fig.4-3 necesită tensiune de contrast negativă. Biblioteca ce configurează modulul de afișare este următoarea:

```
-- filă           : f877_lcd.jal
-- scop           : hd44780  pini IO
-- pini           : vezi tabelul
-- utilizat de    : hd447808
-- important      : se anulează linia include hd44780p din hd44780.jal
```

```
--        HD44780       16f877
-- ----------------------------------
--        1 Gnd         12,31 GND
--        2 Vcc         11,32 VCC
--        3 Contrast
--        4 RS D/I_       9 E1
--        5 R/W_        12,31 GND
--        6 E           10 E2
--        7 D0          19 D0
--        8 D1          20 D1
--        9 D2          21 D2
--       10 D3          22 D3
--       11 D4          27 D4
--       12 D5          28 D5
--       13 D6          29 D6
--       14 D7          30 D7

 var volatile bit  hd44780_8_DI is pin_e1
 var volatile bit  hd44780_8_E  is pin_e2
 var volatile byte hd44780_8_D  is port_d

 procedure _hd44780_8_init is
   port_d               = 0
   pin_e1               = low
   pin_e2               = low
   port_d_direction     = all_output
   pin_e1_direction     = output
   pin_e2_direction     = output
 end procedure
```

Utilizarea comenzilor de scriere în modul se face identic cu cele pentru modul de conexiune 4+2.

4.4 Fantezii de interfaţare pentru micşorarea numărului de pini utilizaţi

De cele mai multe ori, modulele inteligente destinate afişării unei mărimi fizice sau alfanumerice de uz general, utilizează şi o mulţime de butoane necesare fie navigării prin meniuri, fie resetării sau pur şi simplu unor comenzi individuale (schimbarea domeniului de măsură, a rezoluţiei de afişare etc.). Un utilizator avizat va folosi cel mai ieftin microcontroler care se potriveşte aplicaţiei sale sub aspectul numărului de pini utilizaţi şi a memoriei ocupate; de aceea salvarea a 4 sau 8 pini de uz general (IO) este uneori un deziderat important. Exemplul următor arată o soluţie mai puţin ortodoxă de interfaţare a 4 butoane pe aceiaşi pini pe care se conectează modulul LCD. Se observă modul ciudat de conectare al butoanelor S2…S4 între liniile de date şi pinul ***EN****able* al afişajului LCD. Aceasta pentru că în funcţionare normală, HD44780 are nevoie de EN în stare *high* sau de tranziţia EN din stare *high* în stare *low* pentru unele afişaje LCD, în timp ce butoanele au nevoie de semnalul EN în stare *low*, deorece prin setarea registrului OPTION au fost conectate cu rezistenţă de pull-up toate intrările portului B. Apăsarea oricărui buton S2…S4 cât timp EN este high nu are nici un efect, pe deoparte datorită diodelor D2…D4 care nu

conduc şi pe de altă parte datorită direcţiei pinilor Rb0…Rb3 care sunt ieşiri. Când EN este *low* şi Rb0…Rb3 sunt intrări, apăsarea oricărui buton va trece intrarea corespunzătoare din stare logică *high* în stare logică *low*. Utilizatorul nu are altceva de făcut decât să valideze citirea.

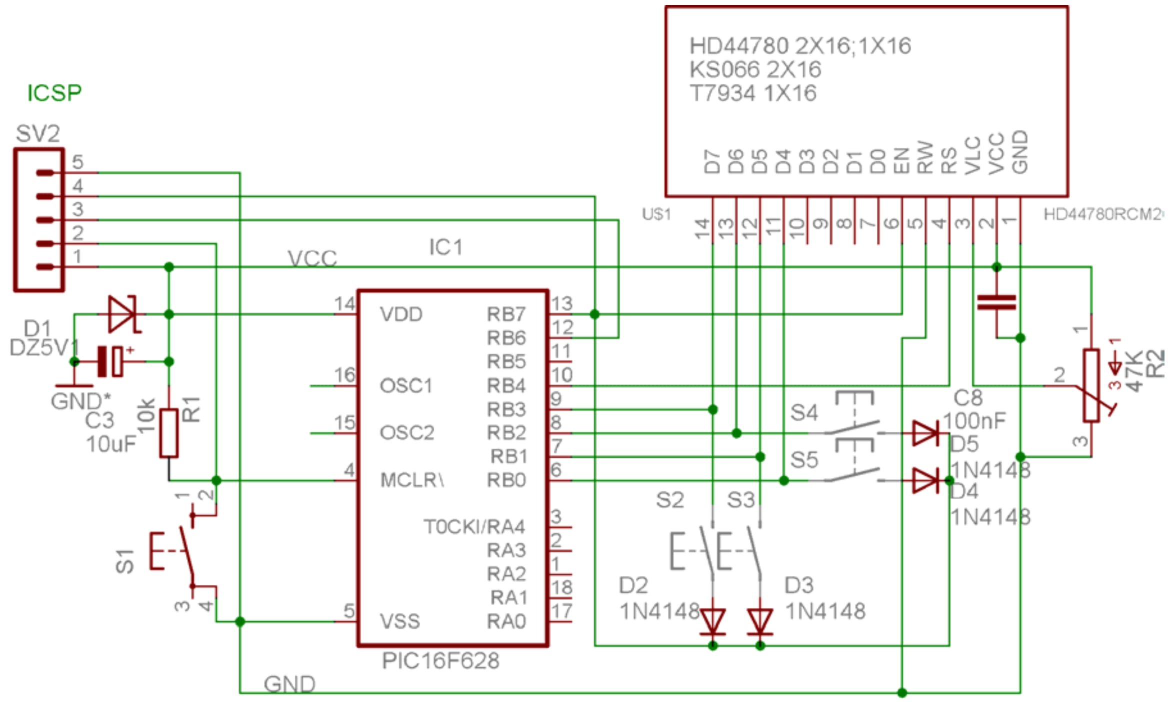


fig.4- 4 Butoane şi LCD utilizând aceiaşi pini

De observat că acest mod de interfaţare nu permite citirea butoanelor în întreruperi generate de TMR0, dacă se doreşte utilizarea bibliotecii originale HD44780.jal, deoarece compilatorul nu permite apelarea aceleiaşi rutine utilizator atât din rutina de întreruperi cât şi din programul principal (în cazul de faţă rutinele de afişare ale HD44780). Cu toate acestea, programul funcţionează foarte bine în modul indicat mai jos:

```
var volatile byte button_direction is port_b_high_direction
var volatile bit buton_1  ; pin_b7
var volatile bit buton_2  ; pin_b6
var volatile bit buton_3  ; pin_b5
var volatile bit buton_4  ; pin_b4

pragma target fuses 0b_0011_1111_0011_1000
;  cp off, lvp off, boden on, mclr, pwrte on, intrc_io, wdt off
cmcon = 0x_07  ; dezactivează comparatoarele
clear_watchdog
option = 0b_0100_1000
--  setează pullup pe port_b, prescaler-ul este asignat wdt-ului, int/rb0 pe front crescător
tmr0 = 0
procedure buton_clear is
-- ----------------------
 buton_1 = low buton_2 = low buton_3 = low buton_4 = low
```

```
end procedure
procedure buton_read is
-- --------------------
hd44780_4_E = low                ; dezactivează lcd
 button_direction = all_input   ; activează butoanele
 if    (! pin_b7 & tmr1if) then buton_1 = on tmr1if = low
 elsif (! pin_b6 & tmr1if) then buton_2 = on tmr1if = low
 elsif (! pin_b5 & tmr1if) then buton_3 = on tmr1if = low
 elsif (! pin_b4 & tmr1if) then buton_4 = on tmr1if = low
 end if
 button_direction = all_output ; dezactivează butoanele
end procedure

-- main program
-- ************
forever loop
; afişează ceva cu rutinele HD44780
 buton_clear
 buton_read
; program utilizator
end loop
```

Se observă trei particularități ale programului:

- Registrul OPTION are bitul de pull-up setat (şi bitul de întreruperi care este folosit aici în alt scop este deasemenea setat)
- După fiecare afişare pe HD44780, se apelează procedura de ştergere a variabilelor buton_x, urmată de buton_read. Omiterea acestei secvențe va duce la setarea aleatoare a variabilelor buton_x, la prima citire, în funcție de data afişată anterior pe LCD.
- Se utilizează PIC16F628 cu oscilatorul intern RC, comparatorul intern este dezactivat

4.5 Principiul serializării

Principiul serializării în microcontrolere se bazează pe existența instrucțiunilor de rotire: ***r**otate **l**eft through **c**arry* respectiv ***r**otate **r**ight through **c**arry.* O rotire la stânga a unui octet înseamnă o înmulțire cu 2 în timp ce o rotire spre dreapta a aceluiaşi octet înseamnă o împărțire cu 2. In Jal înmulțirea şi împărțirea consumă mai mult timp de procesor decât rotirile care sunt native şi apar ca instrucțiuni assembler. Rotirea se execută prin bitul *carry.* De aceea, dacă se lucrează în assembler este necesară setarea sau resetarea acestui bit după o rotire, după cum situația o cere. In Jal acest lucru se face automat prin utilizarea instrucțiunii >> sau <<. Rotirea unui octet se execută în registrul respectiv, deci pentru a vedea efectul acestei rotiri la ieşirea unui pin al microcontrolerului, se poate testa starea bitului carry şi efectuarea unei copii a acestuia pe pinul de ieşire. O rotație completă de 8 ori a orcărui octet, va transfera la un pin de ieşire, bit cu bit, valoarea octetului respectiv. Dacă a avut loc o rotire spre stânga, primul bit serializat va fi cel mai semnificativ (msb), dacă a avut loc o rotire spre dreapta, primul bit serializat va fi cel mai puțin semnificativ (lsb). Această serializare se poate aplica pe intrarea de date a unui registru de deplasare cu încărcare serială şi ieşire paralelă (74LS164, 74HC595, HEF4049), rezultatul fiind refacerea octetului în cauză. Desigur că e nevoie de un semnal de tact ce trebuie generat de un alt pin al microcontrolerului, sincron cu bitul de date transmis. In mod

reciproc, utilizând un registru cu încărcare paralelă şi ieşire serială (74LS165, 74LS166), un octet sau un cuvânt format din unul sau mai mulţi octeţi poate fi citit de microcontroler în mod serial, cu aceiaşi pini de tact respectiv de date folosiţi pentru ieşire.

4.5.1 Interfaţarea LCD prin serializare

Teoria fiind cunoscută şi din parcurgerea subcapitolului 2.9.21, nu avem decât să prezentăm o variantă din multiplele posibilităţi de interfaţare (fig.4-5), ce utilizează un registru de deplasare [12] de 8 biţi, cu încărcare serială pe una din intrările A sau B, (intrarea neutilizată A [sau B] a registrului IC1 având rol de *enable* pentru intrarea de semnal B [sau A]) şi reset asincron activ pe nivel *low*. In aplicaţia prezentată [5], ambele intrări sunt conectate împreună, funcţia *enable* a registrului fiind anulată. Datele sunt rotite la fiecare tranziţie *low-high* a impulsului de tact, dinspre QA spre QH. Modul de conectare al LCD-ului este de fapt 4 + 2, reţeaua R1, D1 asigură generarea semnalului EN printr-o logică de tip *and.* Atât timp cât QH este *high*, dioda D1 nu poate conduce, potenţialul EN fiind fixat *high* prin rezistenţa R1, polarizată de către semnalul de date. Conexiunile registrului la PIC sunt fixate în fila hd44780s_p.jal:

```
-- file       : hd44780s_p.jal
-- used by    : hd447804, în mod serial
var volatile bit hd44780_S_clock is pin_b6
var volatile bit hd44780_S_data is pin_b7
var volatile bit hd44780_S_clock_dir is pin_b6_direction
var volatile bit hd44780_S_data_dir is pin_b7_direction

procedure _hd44780_s_init is
  hd44780_S_clock_dir = output
  hd44780_S_data_dir = output
end procedure
```

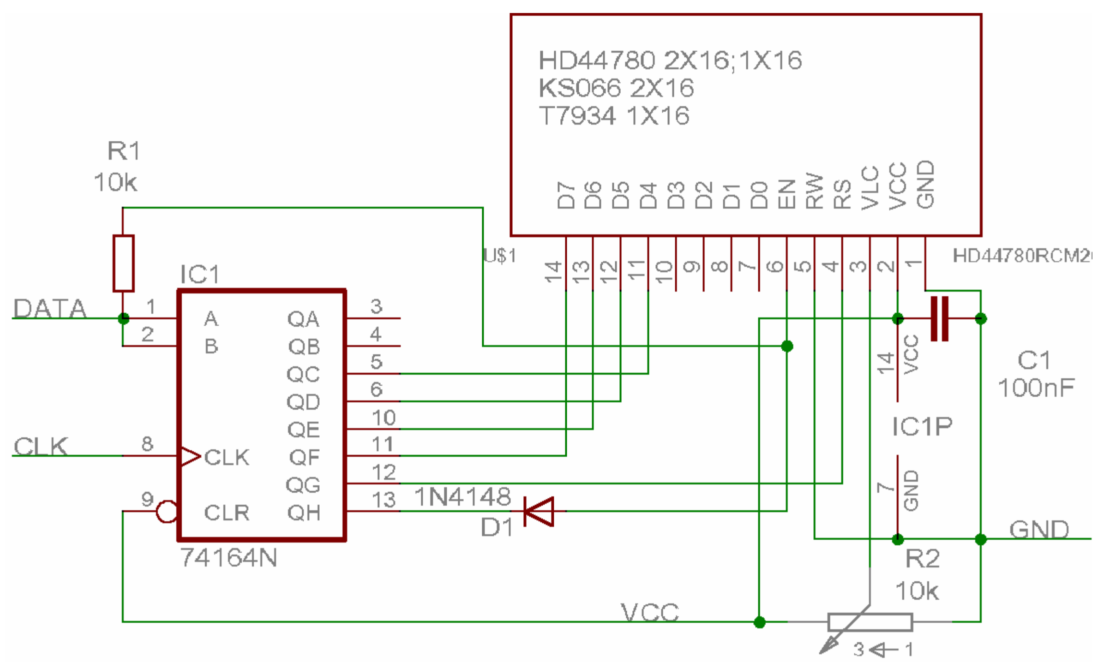


fig.4- 5 Serializarea comenzii unui afişaj LCD nativ-paralel

Biblioteca ce este utilizată de schema electronică din fig.4-5 este următoarea:

```
-- file      : hd44780s.jal
-- purpose   : hd44780 interfaţă serială 74164
-- includes  : jpic, jdelay

include hd44780p
include jpic
include jdelay

procedure send(byte in stemp) is  ; încarcă un octet în 74164
  for 8 loop                      ; execută de 8 ori până la terminarea octetului
    asm bcf hd44780_S_data
    if ((stemp & 128) != 0) then
                                  ; 0b_1000_0000, testează bitul 8 (LCD enable)
      asm bsf hd44780_S_data      ; asigură enable LCD prin semnalul data
    end if
     asm bsf hd44780_s_clock  ;  hd44780_S_clock = high, tranziția high-low
     asm bcf hd44780_s_clock  ;  hd44780_S_clock = low
     asm rlf stemp, f
               ;  stemp = stemp << 1, rotire spre stânga, un pas, MSB este primul bit
  end loop
end procedure
```

-- trimite un octet cu instrucțiunea conținută în variabila value, spre HD44780 :

```
procedure HD44780_instruction( byte in value ) is
  var byte itemp
  itemp = (value >> 2) + 128
                   -- rotire dreapta două poziții şi adunare cu 0b_1000_0000
  send(0)
                   -- şterge registrul SN74164 prin înscriere cu 0
  send(itemp)      -- trimite instrucțiunea
  asm bsf hd44780_s_data    ; hd44780_S_data = high
  asm bcf hd44780_s_data    ; hd44780_S_data = low
  itemp = (value << 2) | 128
                   -- rotire stânga două poziții , sau cu 0b_1000_0000
  send(0)
  send(itemp)
  asm bsf hd44780_s_data    ; hd44780_S_data = high
  asm bcf hd44780_s_data    ; hd44780_S_data = low
end procedure
```

-- trimite data conținută în variabila *value* spre HD44780:

```
procedure HD44780_write( byte in value ) is
  var byte itemp
  itemp = (value >> 2) + 192
                   -- 0b_1100_0000 se adună cu value rotită 2x la dreapta
  send(0)
  send(itemp)
  asm bsf hd44780_s_data    ; hd44780_S_data = high
```

```
  asm bcf hd44780_s_data                      ; hd44780_S_data = low
  itemp = (value << 2) | 192
  send(0)
  send(itemp)
  asm bsf hd44780_s_data                      ; hd44780_S_data = high
  asm bcf hd44780_s_data                      ; hd44780_S_data = low
end procedure

procedure HD44780'put( byte in value ) is -- idem, pseudo-variabilă
  hd44780_write(value)
end procedure

procedure hdinit is                           ; inițializare
_hd44780_s_init                               ; setarea pinilor IO
delay_10mS(2)                                 ; întârziere de inițializare
send(0)
send(140)                                     ; (0x30 >> 2) + 128 = 140
asm bsf hd44780_s_data                        ; hd44780_S_data = high
asm bcf hd44780_s_data                        ; hd44780_S_data = low
delay_1mS(5)                                  ; întârziere de 5mS
asm bsf hd44780_s_data                        ; hd44780_S_data = high, reset
asm bcf hd44780_s_data                        ; hd44780_S_data = low
delay_10uS(16)
asm bsf hd44780_s_data                        ; hd44780_S_data = high, reset
asm bcf hd44780_s_data                        ; hd44780_S_data = low
delay_10uS(16)
send(0)
send(136)
                   ; (0x20 >> 2) + 128  = 136, setează LCD-ul în mod 4-biți
asm bsf hd44780_s_data                        ; hd44780_S_data = high
asm bcf hd44780_s_data                        ; hd44780_S_data = low
delay_10uS(16)
hd44780_instruction(0x28)         -- LCD cu două linii,mod 4-biți
delay_10uS(16)
hd44780_instruction(0x08)         -- stinge afişajul
hd44780_instruction(0x01)         -- şterge DD RAM
delay_1mS(5)
hd44780_instruction(0x06)         -- setează direcția cursorului
hd44780_instruction(0x0c)         -- afişaj aprins, cursor off, blink off
end procedure

hdinit
include hd44780                   -- include  procedura standard
```

Avantajul metodei constă în utilizarea a numai două linii de comandă din microcontroler. Dezavantajul este dimensiunea aproape dublă a codului hexa generat (PIC16F87x, compilator 04.55) pentru afişarea aceluiaşi mesaj "hello!" comparativ cu modul paralel 4 + 2. Dimensiunea este aceeaşi pentru PIC-uri de 1K. Este necesar şi un mic surplus hardware constând în registrul SN74164. Nu se pot utiliza cele patru linii de date ale registrului decât pentru ieşiri, acesta fiind unidirecțional, însă rămân două linii nefolosite (QA şi QB) a căror destinație poate fi semnalizarea optică sau comenzi de ieşire suplimentare.

4.5.2 Interfațarea butoanelor şi a LED-urilor prin serializare

Am văzut în subcapitolul ce tocmai s-a încheiat cât de elegantă este metoda transformării unui afişaj LCD inteligent cu acces paralel, în unul cu acces serial. Realitatea este uşor mai complicată, editarea greşită a unei comenzi care nu generează eroare de compilare, sau montarea inversată a diodei D1, va produce bătăi de cap nebănuite utilizatorului.

Aceeaşi metodă de serializare poate fi folosită cu succes şi pentru citirea stării unui set de 8 butoane. Precauția suplimentară va consta în faptul că anularea efectului tranzițiilor parazite la comutarea acestora trebuie făcută prin metode hardware (trigger schmitd, etc) sau prin metodele software deja prezentate. Metoda se pretează determinării stării unor jumperi sau butoane a căror stare nu se modifică deseori în timpul citirii lor. Schema electronică din fig.4-6 respectă setările originale din biblioteca ciop.jal şi de aceea sunt utilizați trei pini ai microcontrolerului pentru fiecare registru de intrare IC2, respectiv de ieşire IC1, deşi s-ar fi putut utiliza foarte bine acelaşi pin pentru generarea tactului, deoarece acțiunea asupra dispozitivului periferic este secvențială. Numărul de pini necesari pentru interfațare poate fi condensat la doar patru, trei din ei pentru clock, data şi load, comuni ambilor regiştrii IC2, IC3 şi unul pentru **O**utput **E**nable IC1 (în schemă notat ca G), cu condiția ca doar unul dintre regiştrii să aibă ieşirea/intrarea activă la momentul în care se realizează comunicația. Registrul IC1 primeşte informația serială pe pinul SER în avans cu cel puțin 30nS față de tranziția *low-high* a impulsului de tact SCK. După opt tacți, datele sunt încărcate în registru intern de deplasare şi un tact *low-high* pe intrarea RCK transferă datele din registrul intern de deplasare în registrul intern de memorare. Deoarece ieşirea este conectată în permanență, (validarea ieşirii G este menținută permanent în nivel logic *low*) datele memorate sunt transferate spre LED-uri. Este cel mai bun registru cu intrare serială şi ieşire paralelă, deoarece are opțiunea de ieşire cu impedanță ridicată (permite accesul la un bus de date *tri-state*) şi ieşirea este extrem de curată fără *glitch*-uri rezultate la încărcare datorită existenței a doi regiştrii interni de 8 biți, informația trecând din unul în celălalt. Citirea intrărilor registrului IC2 este ceva mai ciudată datorită unei particularități a registrului 74166 [13] de a încărca bistabilii interni cu informația existentă pe intrarea serială SER când SH/LD este în nivel logic *high,* concomitent cu o deplasare a conținutului registrului cu o locație spre dreapta. Această opțiune este necesară pentru cascadarea mai multor astfel de regiştrii. Rezultatul firesc este că încărcarea provoacă şi o deplasare care se pierde în cazul utilizării unui singur registru. Aşadar cei şapte biți de intrare A…G rămaşi, (a căror stare logică este *low* dacă comutatoarele sunt închise, respectiv *high* dacă sunt deschise, intrările TTL fiind în vânt în această situație) sunt transferați sincron cu tranziția *low-high* a semnalului de tact CLK, cu condiția ca SH/LD să revină în prealabil în stare logică *low*. Nu încercați menținerea intrărilor în vânt dacă registrul nu este de tip TTL! Pentru regiştrii CMOS este obligatorie conectarea intrărilor cu rezistențe la VCC.

Pentru testarea circuitului, vă sugerez algoritmul de scriere/depanare al software-ului din aproape în aproape. Este cea mai simplă metodă de a obține un program funcțional cu dimensiuni mari (nu este cazul în exemplul de față). Primul pas va fi studierea bibliotecilor cio.jal şi ciop.jal pentru a vă familiariza cu structura lor şi modificările ce se

impun (numai pentru SN74166). Programul ce verifică transferul datelor pe 74HC595 este format din doar 6 linii:

```
include 16F84_4
include jpic
include cio

cio_out_byte ( 0b_0000_1111 )
cio_out_load
end
```

După ce observaţi într-adevăr modificarea stării logice a LED-urilor ne putem juca cu efecte luminoase pe şirul de 8 LED-uri:

```
; efecte luminoase cu 74HC595
include 16F84_4
include jpic
include jdelay
include cio
```

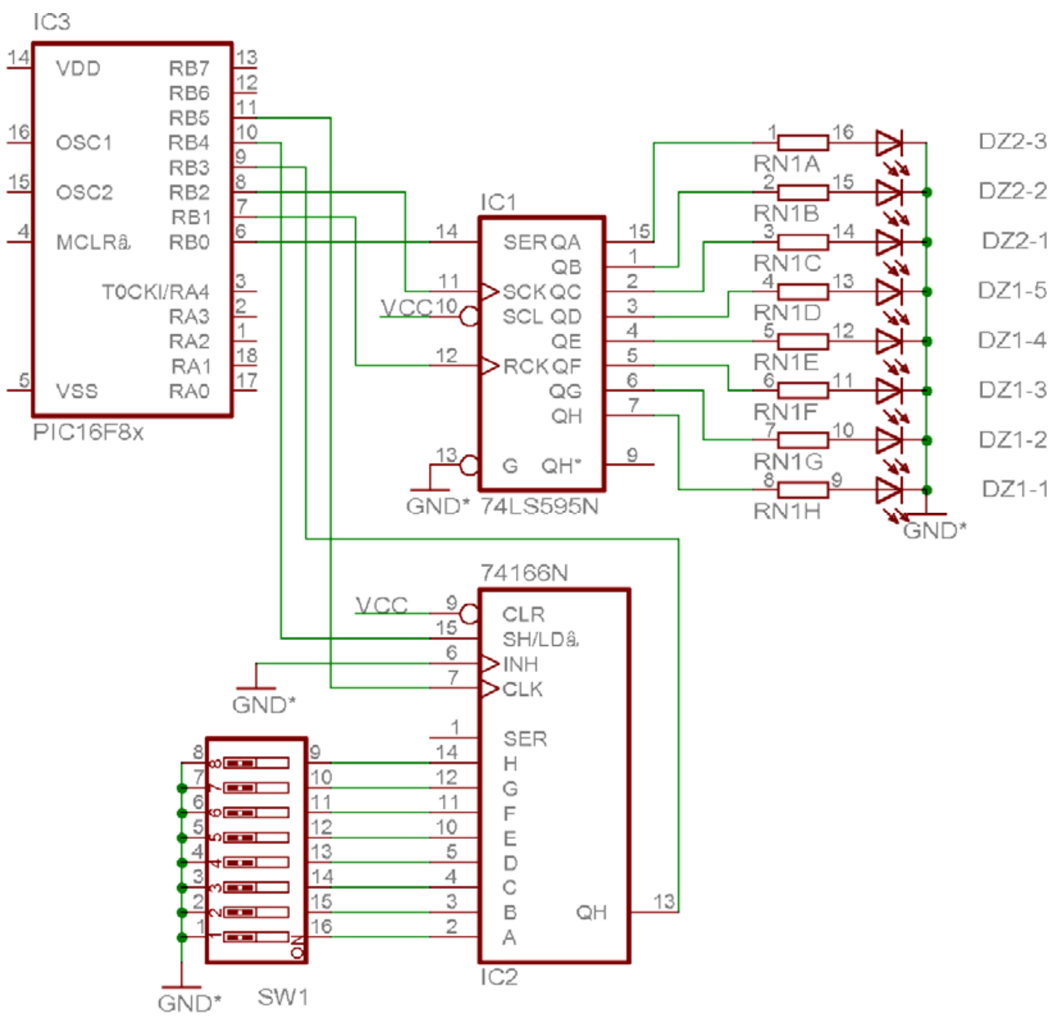


fig.4- 6 Citirea unor micro-întrerupătoare şi afişarea stării acestora prin serializare

```
var byte first = 0b_0000_0001
                                ;  călăreţul porneşte din dreapta cu LED-ul aprins
cio_out_byte ( first )          ;  încărcarea serială a registrului
cio_out_load                    ;  şi scrierea informaţiei pe LED-uri
delay_10mS ( 5 )

forever loop
for 7 loop                      ;  apoi are loc o rotire spre stânga de 7 ori,
 first = first << 1
 cio_out_byte ( first )
 cio_out_load
 delay_10mS ( 5 )
end loop
first = 0b_1000_0000            ;  până se atinge capătul din stânga

 for 7 loop
  first = first >> 1            ;  după care se întoarce spre dreapta, tot de 7 ori
  cio_out_byte ( first )
  cio_out_load
  delay_10mS ( 5 )              ;  delay necesar pentru vizibilitate
 end loop
 end loop                       ;  şi situaţia se repetă la nesfârşit
```

Dacă totuşi vi se pare biblioteca cio.jal prea complicată (este o bibliotecă universală care poate funcţiona cu mai multe tipuri de regiştrii), o variantă a programului care nu o utilizează aproape de loc este prezentată mai jos; de această dată viteza de deplasare a călăreţului este variabilă:

```
include 16f84_4
include jpic
include jdelay

var bit ddata          is pin_b0
var bit load           is pin_b1
var bit clock          is pin_b2
pin_b0_direction = output
pin_b1_direction = output
pin_b2_direction = output

procedure out_byte ( byte in data ) is

 var bit data_bit at data : 0
   for 8 loop                   ;  rotirea octetului de transmis , lsb este primul
      ddata = data_bit
      data = data >> 1
      clock = low               ;  tact low_high pentru incarcare date
      clock = high
   end loop
  load = low                    ;  tact low_high pentru ieşire date
  load = high
end procedure

var byte first = 0b_0000_0001
```

```
var byte viteza = 5          ; viteza reprezintă o cuantă de multiplicare a delay-ului

out_byte ( first )
delay_1mS ( viteza )
forever loop
   for 7 loop
        first = first << 1   ; roteşte octetul de transmis
        out_byte ( first )   ; trimite-l în registru şi afişează-l
        delay_1mS ( viteza )
        viteza = viteza – 1 ; scade delay-ul cu 1mS la fiecare pas
   end loop
first = 0b_1000_0000
       for 7 loop
        first = first >> 1
        out_byte ( first )
        delay_1mS ( viteza )
        viteza = viteza - 1
       end loop
if viteza == 1 then viteza = 5 end if
                             ; dacă e viteza maximă, revine la cea inițială
end loop
```

Se pot imagina variațiuni pe această temă la infinit. Un efect interesant se obține dacă se utilizează LED-uri bicolore (fie cu două, fie cu trei terminale) şi două registre 74HC595. LED-urile bicolore cu două terminale se conectează între ieşirile cu acelaşi nume ale celor două registre prin rezistențe de limitare de cca 330 ohmi conectate în serie cu ele. LED-urile cu trei terminale se conectează cu anozii la ieşirile cu acelaşi nume ale celor doi regiştrii, iar catozii comuni se conectează la masă prin rezistențe de 220…470 ohmi (fig.4-7). Lăsăm la imaginația cititorului modul în care va jongla cu ieşirile regiştrilor, obținând efecte luminoase tricolore intermitente fie la nivel de şir, fie la nivel de LED. Dacă cititorul are de unde să procure LED-uri RGB (sunt ceva mai scumpe) utilizând trei (sau mai mulți regiştrii conectați în mod *daisy-chain,* adică înlănțuiți) pot obține efecte luminoase având aproape orice culoare. Să revenim la citirea microîntrerupătoarelor din fig.4-6. Se impune o mică modificare a uneia dintre rutinele din biblioteca cio.jal (vezi CD:\tools\jal_compiler) după cum urmează:

```
-- input parallel load
procedure cio_in_load is
   _cio_delay
   ; _cio_in_load = on           -- _cio_in_load_active
   _cio_in_load = off            -- !_cio_in_load_active
      if _cio_in_clocked_load then
         _cio_in_pulse_clock
      else
         _cio_delay
      end if
   ; _cio_in_load = off    -- !_cio_in_load_active
   _cio_in_load = on       --  _cio_in_load_active
   _cio_delay
end procedure
```

Este vorba de modificarea polarității pulsului SH/LD care pentru registrul 74166 trebuie să fie activ pe nivel logic *low* pentru încărcare. Programul care citeşte starea comutatoarelor este foarte scurt:

```
; test citirea serială: in 74166, out 74HC595
include 16F628_4
include jpic628
include jdelay
include cio
var byte read_button

forever loop
 cio_in_load ;  încarcă registrul şi execută (nedorit !) o deplasare la dreapta
 cio_in_byte ( read_button )       ;  citeşte registrul în microcontroler
 read_button = read_button >> 1
                                   ;  deoarece a avut loc o deplasare o corectăm
 cio_out_byte ( read_button )      ;  şi o scriem în registru 595
 cio_out_load                      ;  şi apoi pe LED-uri
end loop
```

Doar 7 (intrările A…H ale 74166) din cele 8 microcomutatoare vor fi citite. Pentru a citi şi cel de-al optulea, este nevoie de încă o celulă de bistabil conectată la ieşirea QH a registrului care să memoreze prima rotire nedorită ce are loc odată cu încărcarea registrului.

Microcontrolerul din fig.4-7 este protejat împotriva alimentării inversate, de dioda D1. Alimentarea lui se face pe acelaşi conector ce realizează funcția de programare (SV1, ICSP). De această dată se utilizează şi pinul de validare al încărcării (G) pentru a opri "clipirea" şirului de led-uri la fiecare încărcare serială a registrului.

```
include 12f675_4i ; comentați linia "include jpic" pentru a utiliza
include jpic675   ; biblioteca jpic675
osc_calibrate     ; rutină pentru calibrarea oscilatorului RC intern
gp0_direction = output
gp1_direction = output
gp2_direction = output
gp5_direction = output
gp4_direction = output
var bit red_data       is gp1
var bit enable         is gp2
var bit green_data     is gp0
var bit clock          is gp5
var bit load           is gp4
var byte viteza
const bit red = low    ; definirea culorii LED-ului accesat de registru
const bit green = high
procedure out_byte ( byte in data, bit in color ) is
enable = on     ; 595 trece in tri-state pe perioada înscrierii cu date, astfel LED-urile nu pâlpâie
 var bit data_bit at data : 0
   for 8 loop
     if color == red then
      red_data = data_bit
      green_data = low
     elsif color == green then
```

```
      green_data = data_bit
      red_data = low
     end if
      data = data >> 1
      clock = low
      clock = high
   end loop
```

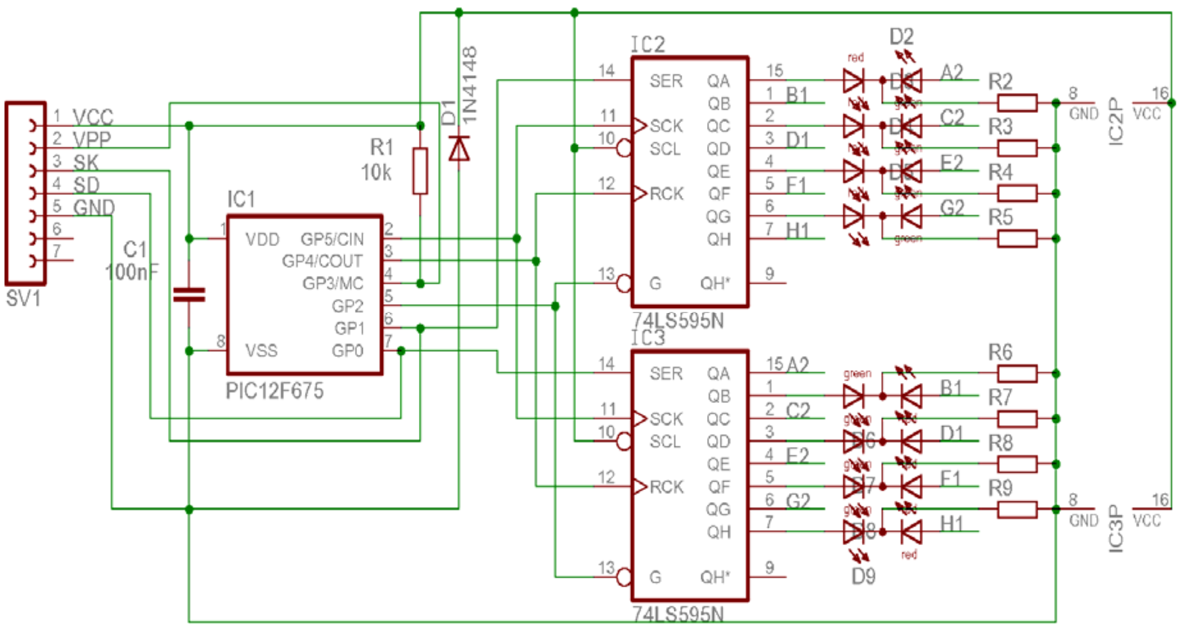


fig.4- 7 Efecte luminoase bicolore (roşu, verde) cu PIC12F675 şi 74HC595

```
load = off
load = on
enable = off
end procedure
viteza = 5
forever loop ; secvenţă de afişare verde-roşu conform valorii încărcate în regiştrii
 out_byte ( 0, red )     out_byte ( 0b_1000_0000, green )
 delay_100mS ( viteza ) out_byte ( 0b_1100_0000, green )
 delay_100mS ( viteza ) out_byte ( 0b_1110_0000, green )
 delay_100mS ( viteza ) out_byte ( 0b_1111_0000, green )
 delay_100mS ( viteza ) out_byte ( 0, green )
 out_byte ( 0b_0000_1000, red ) delay_100mS ( viteza )
 out_byte ( 0b_0000_1100, red ) delay_100mS ( viteza )
 out_byte ( 0b_0000_1110, red ) delay_100mS ( viteza )
 out_byte ( 0b_0000_1111, red ) delay_100mS ( viteza )
end loop
```

4.6 Conversia AD



Conversia **A**nalogic – **D**igital reprezintă unul din cele mai spectaculoase aspecte situate la graniţa dintre electronica pur analogică şi electronica digitală. Este interesantă deoarece utilizatorul trebuie să stăpânescă la perfecţie problemele pe care le ridică măsurarea semnalelor analogice de nivel mic, în prezenţa zgomotelor introduse de comutaţia sistemului digital. Problemele sunt cu atât mai spinoase cu cât numărul de biţi al convertorului interfaţat este mai mare, (18 …20 biţi) deci rezoluţia de măsură este în domeniul microvolţilor. Acest capitol nu-şi propune să trateze în detaliu teoria funcţionării convertoarelor AD şi particularităţile acestora, ci doar problemele specifice ale convertorului AD cu eşantionare şi memorare conţinut în seria PIC16F87x respectiv interfaţarea câtorva convertoare de 12, 14 şi 18 biţi, clasice la ora actuală. Cu toate acestea, o clasificare rapidă a convertoarelor AD după principiul de funcţionare poate fi următoarea:

- Metode hardware:
  - conversie tensiune frecvenţă sau tensiune timp
  - conversie simplă pantă, dublă pantă sau multiplă pantă
  - conversie cu eşantionare-memorare (sample & hold)
  - conversie delta-sigma

- Metode software:
  - aproximaţia succesivă
  - măsurarea timpului de încărcare sau de descărcare al unui condensator

Cuvintele cheie ce definesc orice convertor AD sunt: rezoluţie, neliniaritate, monotonie, cap de scală (*full-scale*), eroare de ofset (*zero scale ofset*), impedanţă de intrare, viteză de răspuns sau timp de conversie, metodă de comunicaţie (serială sau paralelă). Se pot implementa convertoare AD utilizând convertoare DA şi metode hardware sau software. Pentru a vorbi aceeaşi limbă cu cititorul, este importantă definirea cuvintelor cheie amintite mai sus:

- **rezoluţia** este numărul maxim de stări logice distincte la ieşire, pentru o tensiune de intrare:

  $\mathbf{r_d = 2^n}$ sau $\mathbf{r_d = n}$ (exprimare în număr de biţi) sau
  $\mathbf{r_a = FS/2^n}$ (exprimare în unităţi analogice)

De exemplu: un convertor de 10 biţi va avea $2^{10} = 1024$ de stări distincte pentru o tensiune egală cu capul de scală, dacă aceasta este 5V atunci rezoluţia convertorului exprimată în unităţi analogice va fi: $r_a = 5V/1024 = 4.8mV$.

Pentru un convertor de 18 biţi rezoluţia exprimată în unităţi analogice pentru acelaşi cap de scală de 5V va fi: $r_a = 5V/2^{18} = 19uV$, respectiv rezoluţia exprimată în unităţi digitale este: $r_d = 2^{18} = 262144$

- **cap de scală** (**F**ull **S**cale) este cea mai mare valoare posibilă la ieşirea convertorului AD. O variaţie a mărimii analogice de intrare peste valoarea maximă corespunzătoare FS va avea efect nul asupra ieşirii digitale a convertorului. Eroarea capului de scală reprezintă dispersia FS a lotului de convertoare de la valoarea măsurată, raportată la valoarea ideală a ieşirii, şi se măsoară în fracţiuni de LSB.

- **eroarea de ofset** este valoarea digitală de ieşire, corespunzătoare unei tensiuni de intrare nule. Se poate exprima în fracţiuni de LSB, **p**ărţi **p**e **m**ilion, fracţiuni de FS, sau unităţi de măsură analogice (mV).

- **neliniaritatea diferenţială** este diferenţa între deviaţia maximă a mărimii de ieşire pentru două stări succesive în intrare şi deviaţia ideală corespunzătoare reprezentată de ecuaţia unei linii drepte. Se măsoară în fracţiuni de LSB. De exemplu ± ½ LSB înseamnă o deviaţie de ieşire cuprinsă între 1 – ½ LSB şi 1 + ½ LSB pentru două valori succesive ale intrării analogice ce produc o modificare a ieşirii digitale.

- **neliniaritatea integrală** este deviaţia maximă a mărimii de ieşire faţă de linia dreaptă trasată prin punctele extreme ale caracteristicii convertorului.

- **monotonia** este proprietatea convertorului de a avea o variaţie pozitivă sau cel puţin nulă pentru o variaţie pozitivă a mărimii analogice de intrare.

- **impedanţa de intrare** (sau impedanţa maximă admisă a sursei de semnal) reprezintă raportul $\Delta U/\Delta I$ unde U este tensiunea de intrare, iar I este curentul absorbit de intrare în condiţiile cele mai dificile de variaţie a temperaturii mediului ambiant. Poate să nu fie definită explicit ci implicit, ca o limită a impedanţei maxime de ieşire a sursei de semnal conectate pe intrarea convertorului.

- **timpul de conversie** este intervalul de timp necesar generării codului binar din momentul startării conversiei şi poate avea diverşi substituenţi ca perioada de conversie AD sau timp de achiziţie, în funcţie de producătorul convertorului şi modul lui de funcţionare.

4.6.1 Utilizarea modulului AD intern al PIC16F87x, biblioteca analogică

Microcontrolerul PIC16F87x dispune de un convertor de 10 biți cu 5 sau 8 canale în funcție de tipul de capsulă (cu 28 respectiv cu 40/44 de pini). Este un convertor de uz general, suficient de precis pentru o multitudine de aplicații, pornind de la măsurarea tensiunii sau curentului în aplicații industriale şi terminând cu măsurarea semnalelor bioelectrice (utilizând amplificatoare corespunzătoare) în medicină. Descrierea accesului utilizatorului asupra modului de conversie este prezentat în fig.4-8 respectiv [14] CD:\datasheet\microchip\pic16F87xx, secțiunea "Analog-to-Digital converter module".

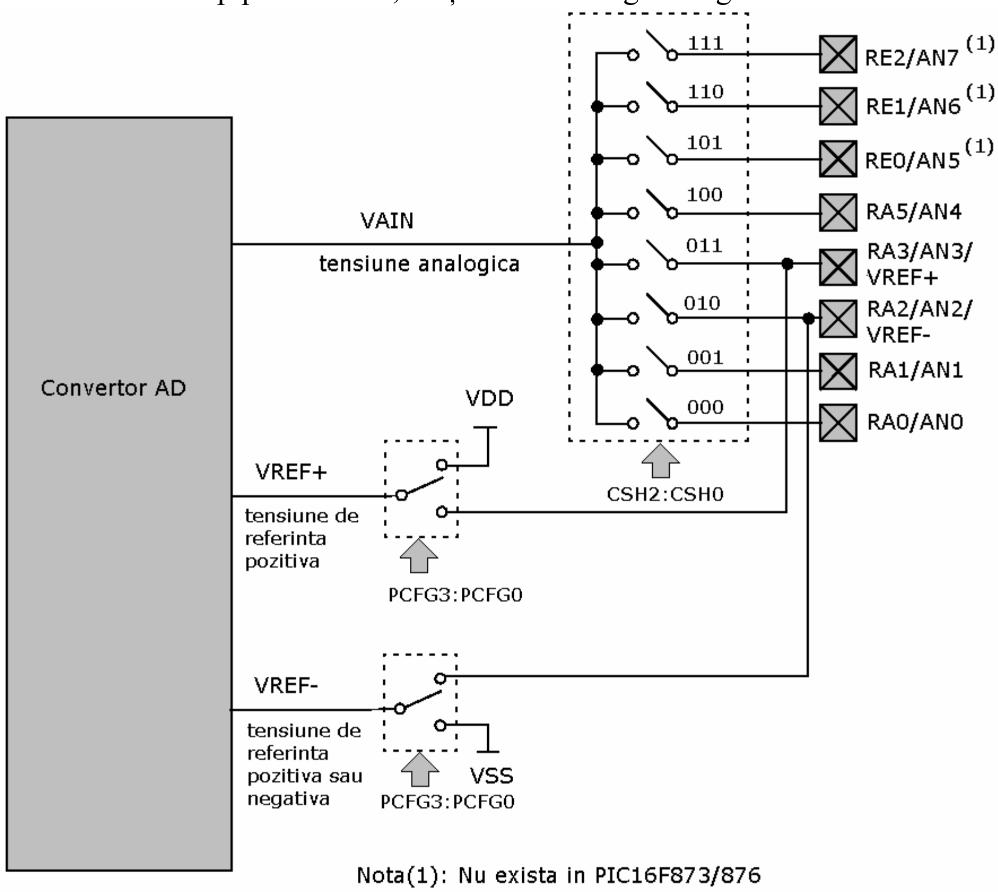


fig.4- 8 Etajul de multiplexare din convertorul AD al seriei PIC16F87x

Un simplu multiplexor analogic comandat de biții CHS0…CHS2 din registrul ADCON0, setează canalul analogic destinat măsurării, în timp ce biții PCFG3…PCFG0 din registrul ADCON1 setează cele două tensiuni de referință, fie intern (+VREF ≤ VDD respectiv –VREF = VSS), fie extern pe canalele RA2 respectiv RA3, care devin atunci intrări de referință şi nu mai pot fi utilizate pentru măsurarea tensiunii. Deoarece setarea canalului destinat achiziției este relativ dificilă conform tabelului din fig.4-9, am imaginat o bibliotecă care să poată fi utilizată fără a avea nevoie de acest tabel şi să conțină toate variantele de utilizare a convertorului AD. Deoarece biblioteca are o dimensiune destul de mare şi se găseşte integral pe CD, voi prezenta doar rutinele necesare înțelegerii exemplului de achiziție implementat.

ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0
R/W	U	U	U	R/W	R/W	R/W	R/W

ADFM: bitul de selecție al formatului rezultatului:
1 = aliniere la dreapta, cei mai semnificativi 6 biți ai ADRESH sunt citiți 0
0 = aliniere la stânga, cei mai puțin semnificativi 6 biți ai ADRESL sunt citiți 0
PCFG3:PCFG0: biții de configurare ai convertorului AD
Notă:
R/W = read/write; U = neimplementat; A = analogic, D = digital
* canalele AD nu sunt disponibile pentru PIC16F876/873 (28 pini)

PCFG 3:0	AN7 RE2*	AN6 RE1*	AN5 RE0*	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	Vref+	Vref-	AD /ref
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	Vref+	A	A	A	RA3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	Vref+	A	A	A	RA3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	Vref+	D	A	A	RA3	VSS	2/1
011X	D	D	D	D	D	D	D	D	VDD	VSS	0/0
1000	A	A	A	A	Vref+	Vref-	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	Vref+	A	A	A	RA3	VSS	5/1
1011	D	D	A	A	Vref+	Vref-	A	A	RA3	RA2	4/2
1100	D	D	D	D	Vref+	Vref-	A	A	RA3	RA2	3/2
1101	D	D	D	D	Vref+	Vref-	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	Vref+	Vref-	D	A	RA3	RA2	1/2

fig.4- 9 Registrul ADCON1, responsabil cu selecția canalului de măsură

```
-- file      : janalog.jal
-- date      : septembrie 2000, modificat martie 2001
-- purpose   : citirea mărimii analogice cu F87x
-- requires  : jpicm.jal, minim jal04.24
-- ---------------------------------------------------------
-- următoarele rutine sunt disponibile aici:
-- ch0_on; ch1_on; ch2_on; ch3_on; ch4_on; ch5_on; ch6_on; ch7_on;
-- ad1_noref; ad3_noref; ad5_noref; ad6_noref; ad8_noref
-- ad2_refplus; ad4_refplus; ad5_refplus; ad7_refplus
-- ad1_refboth; ad2_refboth; ad3_refboth; ad4_refboth; ad6_refboth
-- no_ad; ch_write, ch_write_1023
-- sunt utilizate toate posibilitățile de lucru cu AD-F87x
-- ---------------------------------------------------------

include bin2bcd3
include jdelay
var byte msd, isd, lsd
procedure ch1_on is   ; inițializarea conversiei pe canalul a1
    if target_clock <= 10_000_000 then
```

```
        f877_adcon0 = 0b_0100_1001          -- a1, ad on, TAD = 8TOSC
     elsif target_clock > 10_000_000 then
        f877_adcon0 = 0b_1000_1001          -- a1, ad on, TAD = 32TOSC
     end if
  end procedure

  procedure ch2_on is ; iniţializarea conversiei pe canalul a2
     if target_clock <= 10_000_000 then
        f877_adcon0 = 0b_0101_0001          -- a2, ad on
     elsif target_clock > 10_000_000 then
        f877_adcon0 = 0b_1001_0001
     end if
  end procedure

procedure ad2_refboth is
; setarea convertorului cu două referinţe externe şi două canale de măsură: a0 şi a1
    bank_1
    f877_adcon1 = 0b_1000_1101
                          -- a0,a1 adinput, a4,a5,a6,a7 digital IO, right justified
    bank_0
 end procedure

var byte ch_hi = 0
var byte ch_lo = 0

procedure ch_write is
-- -----------------------------------
  delay_10uS ( 2 )                    -- aşteaptă timpul minim de achiziţie
  adcon0_go = high                    -- start conversie
  while adcon0_go loop end loop       -- aşteptă terminarea conversiei
  ch_hi = f877_adresh                 -- copiază f877_adresh în ch_hi
  bank_1
  asm  movf    f877_adresl,w          -- f877_adresl este în bank_1, mută-l în w
  bank_0
  asm  movwf   ch_lo                  -- şi apoi în ch_lo aflat în bank_0
end procedure

procedure ch_write_1023 is
-- ------------------------------------
-- returnează valoarea zecimală a conversiei în format 0…1023
  ch_write
  bin2bcd3 ( msd, isd, lsd, ch_hi, ch_lo )
                                     -- procedură de conversie 2 octeţi în 3 bcd
 end procedure
; notă asupra prescurtărilor de mai sus:      most significant digit,
;                                             intermediate significant digit,
;                                             last significant digit
```

Procedura *ch_write* returnează în regiştrii lsd respectiv isd, valoarea hexadecimală a conversiei, aliniată la dreapta (*right justified*) datorită înscrierii valorii *high* în bitul ADFM din registrul ADCON1 (fig.4-9). Prin această aliniere la dreapta va rezulta valoarea reală a conversiei:

	ch_hi	ch_lo	zecimal
Right justified	0b_0000_0011	0b_1111_1111	1023
Left justified	0b_1111_1111	0b_1100_0000	32736

în timp ce o aliniere la stânga va multiplica rezultatul cu 2^6. Este mult mai simplă obținerea multiplicată a rezultatului prin configurarea unui singur bit decât printr-o operație de multiplicare pe 10 biți (doi octeți implicați în operație). Păcat că producătorul microcontrolerului nu a rezervat doi biți pentru înmulțirea hardware cu un număr variabil cuprins între 2^3 şi 2^6 a rezultatului conversiei; oricum acest lucru poate fi realizat destul de uşor prin software.

Un exemplu simplu de citire cu viteză mică a valorilor analogice aplicate pe intrările AN0 şi AN1, utilizând biblioteca descrisă anterior, este în programul următor:

```
include f877_04
include jpic
include janalog
include jprint
include jdelay
include f877_lcd ; biblioteca de configurare a pinilor conform fig.4-10
include hd447808

hd44780_clear  -- initializare afişaj LCD
ad2_refboth    -- a0,a1 intrări analogice, +vref, -vref active

forever loop
  ch0_on           -- porneşte conversia pe ch_0
  ch_write_1023    -- converteşte rezultatul ch_lo şi ch_hi în format  bcd
   hd44780_line1
    print_hexadecimal_2 ( hd44780, isd, "0" )
    hd44780_position1 ( 2 )
    print_hexadecimal_2 ( hd44780, lsd, "0" )
  ch1_on           -- porneşte conversia pe ch_1
  ch_write_1023    -- execută conversia AD şi transformă rezultatul în format BCD
    hd44780_line2
    print_hexadecimal_2 ( hd44780, isd, "0" )
    hd44780_position2 ( 2 )
    print_hexadecimal_2 ( hd44780, lsd, "0" )
  delay_100mS ( 3 ) -- delay necesar pentru afişare
end loop
```

Schema de aplicație este cea din fig.4-10. Se observă importanța separării masei analogice de cea digitală şi conectarea acestora într-un singur punct având impedanța AC minimă, foarte aproape de conectorul sursei de alimentare. De remarcat că o sursă de alimentare bine proiectată împreună cu un cablaj corect realizat, va face ca impedanța VCC respectiv GND să fie egale şi foarte apropiate de zero. Este vorba despre impedanța de ieşire din sursa de alimentare care este cuprinsă uzual între 0.1 şi 0.3 ohmi, iar impedanța masei este sub 0.1 ohmi, dacă cablajul are secțiunea suficient de mare şi nu se creează bucle parazite de masă. Un condensator suplimentar de filtraj (100nF + 10uF) situat în imediata apropiere a pinilor de alimentare ai microcontrolerului poate reduce semnificativ zgomotele injectate în alimentări de comutația digitală. In fig.4-10 observați o **particularitate pe care nu o veți**

găsi în cartea tehnică a microcontrolerului chiar dacă v-ar ajuta nişte ochi de şoim: tensiunea aplicată pe ANA1 sau ANA2 poate fi negativă, cuprinsă între limitele: -0.5V…+4.5V sau pozitivă, cuprinsă între 0V şi +5V. Pentru a putea măsura tensiuni negative în apropierea lui 0V trebuie ca: -Vref (RA2) = - 0.5V respectiv +Vref (RA3) = +4.5V. Se observă că relaţia: | -Vref| + |+Vref| <= 5V trebuie să rămână valabilă şi pentru: -Vref < 0. Măsurarea tensiunii negative este posibilă şi datorită imperfecţiunii circuitelor de protecţie existente pe fiecare intrare a microcontrolerului, circuite formate din două diode care se deschid la (GND – 0.6V) respectiv la (VCC + 0.6V).
Se observă că limita de -0.5V aplicată orcărei intrări nu deschide încă dioda de protecţie dar este necesară limitarea curentului de măsură cu o rezistenţă serie (R8 pe referinţă respectiv R4 şi R5 pe măsură) pentru a împiedeca depăşirea curentului maxim admis absorbit din intrare, în cazul creşterii acestei valori peste |-0.5V|.

Obţinerea rezoluţiei maxime de măsură (1024 de puncte) se face pentru: |+Vref| - |-Vref| = 2.5V...2.8V. Condiţia este adevărată numai pentru **0 < |-Vref| < +2.5V** şi **+2.5V < +Vref < +5V** . Deci nu încercaţi: -Vref = +4V şi +Vref = +5V pentru că nu va merge decât cu o pierdere însemnată de rezoluţie, în timp ce: -Vref = 2V şi +Vref = 4.5V sau -Vref = 0V şi +Vref = +5V este o opţiune perfect funcţională. Dacă este necesară măsurarea unui semnal mic (sub 1V) singura soluţie rămasă este utilizarea unui amplificator operaţional extern.

Câteva cuvinte trebuie spuse despre impedanţa de ieşire a sursei de semnal. Microchip solicită ca impedanţa echivalentă a sursei de semnal să fie maximum 10 K. Pentru această impedanţă se recomandă un timp de achiziţie de cca. 20 uS, timp provenit din ecuaţia următoare:

TACQ = Amplifier Settling Time + Hold Capacitor Charging Time + Temperature Coefficient
sau:

$TACQ = T_{AMP} + T_C + T_{COFF} = 2uS + T_C + [(temperature – 25C)(0.05uS/C)$
iar:

$T_C =$ CHOLD (RIC + RSS + RS)ln (1/2047) = 120pF (1K +7K +10K) ln (0.0004885) = 16.5uS
unde:

RIC – rezistenţa internă a condensatorului
RSS – impedanţa internă a comutatorului de sampling
RS – impedanţa sursei de semnal
CHOLD – capacitatea tipică a condensatorului de memorare (hold)
TACQ – timpul de achiziţie

$TACQ_{50C} = 2uS + 16.5uS + [(50C – 25C)(0.05uS/ C)] = 19.7uS$

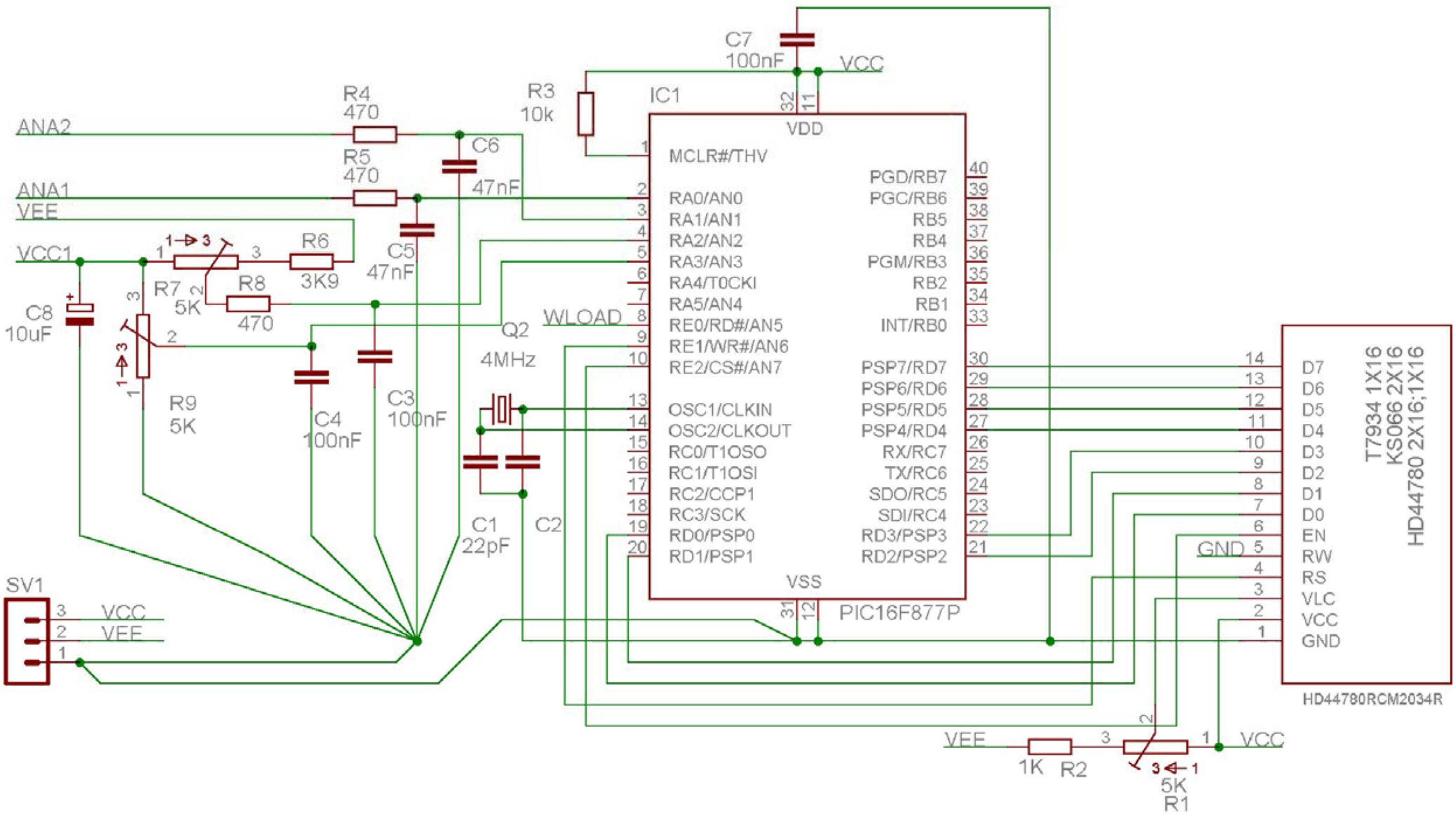


fig.4- 10 Achiziția de date cu polaritate pozitivă şi negativă cu PIC16F877

Limitarea tensiunilor de intrare analogice cu rezistente, fara a scadea dramatic impedanta de iesire a semnalului peste limita solicitata de foaia de catalog (10K) si utilizarea unor tensiuni de referinta alese corect, permit masurarea unor tensiuni negative de valori mici, fara deteriorarea liniaritatii de masura. Conectarea corecta a masei analogice si a masei digitale este de obicei esentiala pentru orice tip de masuratoare analogica cu microcontrolerul PIC. C8 creeaza punctul de impedanta minima al masei analogice.

Modelul analogic al circuitului de eşantionare-memorare la care se referă ecuaţiile anterioare, este cel din fig.4-12. Se observă că un curent de pierderi de cca ±0.5µA poate afecta semnalul de intrare. La 25C acest curent va produce o cădere de tensiune pe rezistenţa sursei de semnal (10K) de aproximativ 50mV. Raportat la capul de scală FS = +5V, eroarea introdusă este de aproximativ 1% ceea ce este încă acceptabil. Deci se pot obţine rezultate bune şi pentru impedanţe de ieşire ale sursei de semnal mai mari decât cele cerute în fila de catalog, cu mărirea corespunzătoare a TACQ şi menţinerea valorilor de intrare a tensiunii cât mai mari. Dacă nu se asigură TACQ şi se măsoară semnale analogice pe mai multe canale, efectul observabil va fi cel de "muşcare" între canale, respectiv variaţia tensiunii pe un canal adiacent va modifica aparent tensiunea achiziţionată pe canalul curent.

Un parametru important este TAD, timpul necesar de conversie pentru fiecare bit convertit. Pentru 10 biţi sunt necesari cel puţin 12TAD. Sunt disponibile patru variante pentru setarea TAD:

Sursa de tact pentru convertorul AD (TAD)		**Frecvenţa maximă [MHz]**
Operaţia	**ADCS1:ADCS0**	
2TOSC	00	1.25
8TOSC	01	5
32TOSC	10	20
RC (nota 1,2)	11	(nota 1)

fig.4- 11 Setarea corectă a biţilor răspunzători de timpul de conversie

Nota 1: Oscilatorul RC are TAD tipic de 4µS dar acesta poate varia între 2-6 µS

2: Când frecvenţa oscilatorului este mai mare de 1MHz, se recomandă utilizarea oscilatorului intern RC numai pentru achiziţie analogică în starea SLEEP

Aceste variante sunt setate prin modificare corespunzătoare a biţilor ADCS1 respectiv ADCS0 din registrul ADCON0, setare realizată în procedurile chX_on din biblioteca janalog.jal. De remarcat că biblioteca janalog.jal nu realizează citirea convertorului AD în întreruperi.

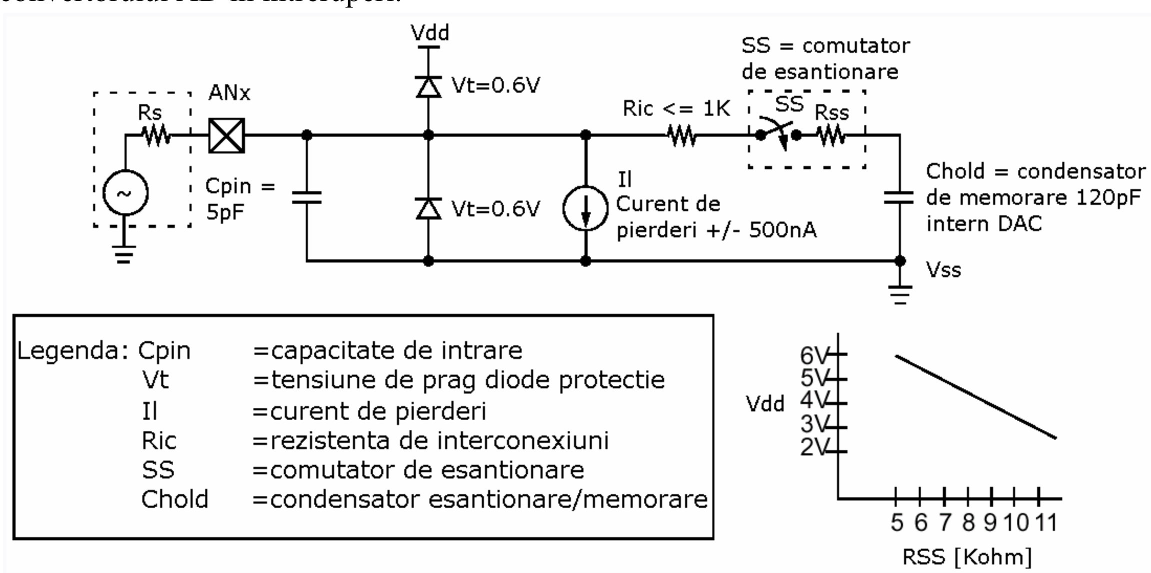


fig.4- 12 Modelul analogic corespunzător intrării AD pentru PIC16F87x

4.7 Convertorul AD de ±18 biți MAX132, exemplu de interfațare

Convertorul AD MAX132 este un convertor performat de 18 biți şi semn, ce funcționează pe baza principiului de integrare cu pantă multiplă (multi-slope), având un FS = ±512mV şi o rezoluție de 2uV/LSB. Poate executa pînă la 100 de conversii /secundă. Consumă cca 60uA în regim de funcționare şi doar 1uA în regim de *sleep*. Interfațarea cu microcontrolerul se realizează în mod serial pe patru fire. La convertoarele cu integrare este important ca durata de integrare a semnalului de referință să fie un multiplu al celui mai puternic perturbator (care este rețeaua de curent alternativ), pentru ca sfârşitul integrării să găsească tensiunea de referință curată şi nu suprapusă peste zgomotul injectat de alimentare. Deoarece la ora actuală există în lume două sisteme de distribuție a tensiunii alternative, respectiv cu frecvența de 50Hz şi 60 Hz, sunt posibile ambele moduri de rejecție a perturbatorilor prin modificarea timpului de integrare fie la 655 perioade de tact, fie la 545 perioade de tact. Oscilatorul ce generează această bază de timp este stabilizat cu un cuarț de 32768 Hz. Numai în situația când nu este necesară rejecția perturbatorilor, rata de citire poate fi crescută la 100 citiri/secundă, cu diminuarea rezoluției convertorului la numai ±13 biți. Rezoluția acestui convertor este definită ca:

Rezoluția [Volți/LSB] = Vin(FS)/262144

Pentru performanțe optime, producătorul recomandă ca FS analog să fie cuprins între ±390mV şi ±550mV pentru o operare în sisteme cu frecvența de 50Hz. Deoarece INHI şi INLO sunt intrări CMOS, protecția acestora la depăşirea accidentală a tensiunii maxime de intrare este foarte importantă şi este realizată prin R1[fig.4.14]. Un filtraj suplimentar al semnalului de intrare este necesar, realizat prin R1, C3. Tensiunea de referință trebuie să aibă valoarea de 655mV iar stabilitatea ei trebuie să fie în mod evident de ordinul ppm (**p**ărți **p**e **m**ilion). Expresia ce determină valoarea referinței pentru un cap de scală dorit este:

$$Vref = \frac{655impulsuri \times 512 \times Vin(FS)}{262144}$$

Domeniul tensiunilor de referință recomandat este 500mV…700mV. Utilizarea altor valori duce la degradarea liniarității convertorului. Cum arată secțiunea analogică din convertor se poate vedea în [fig.4-13] iar schema de aplicație în [fig.4-14]. Etajul de intrare cu comutatoare CMOS execută secvența dictată de interfața digitală, potențialul este repetat printr-un buffer iar apoi integrat cu constanta de integrare dictată de utilizator, un comparator permițând încărcarea unui registru din secțiunea digitală cu un număr proporțional cu tensiunea de intrare (pentru o referință perfect constantă). Rezistența de integrare (R_{INT} fig.4-13, R2 fig.4-14) se determină din:

$$R_{INT} = Vref/I_{INT}$$

Unde I_{INT} = 0.5uA…2.5uA

Iar condensatorul de integrare (C_{INT}, C5) din:

$$C^{INT} = Vin(FS)\ t_{INT}/R_{INT}\ V_{SWING}$$

Unde $V_{SWING} = 1V \ldots 3.5V$
$t_{INT} = 655/f_{OSC}$

Condensatorul de referinţă (C4) depinde doar de frecvenţa oscilatorului : $C_{REF} = 0.0033/f_{OSC}$
Acesta trebuie să aibă curenţi de pierderi cât mai reduşi, deoarece memorează tensiunea de referinţă pe perioada de integrare a acesteia (*deintegrate phase*), un condensator cu dielectric film metalizat (PMP) va fi cea mai bună opţiune.

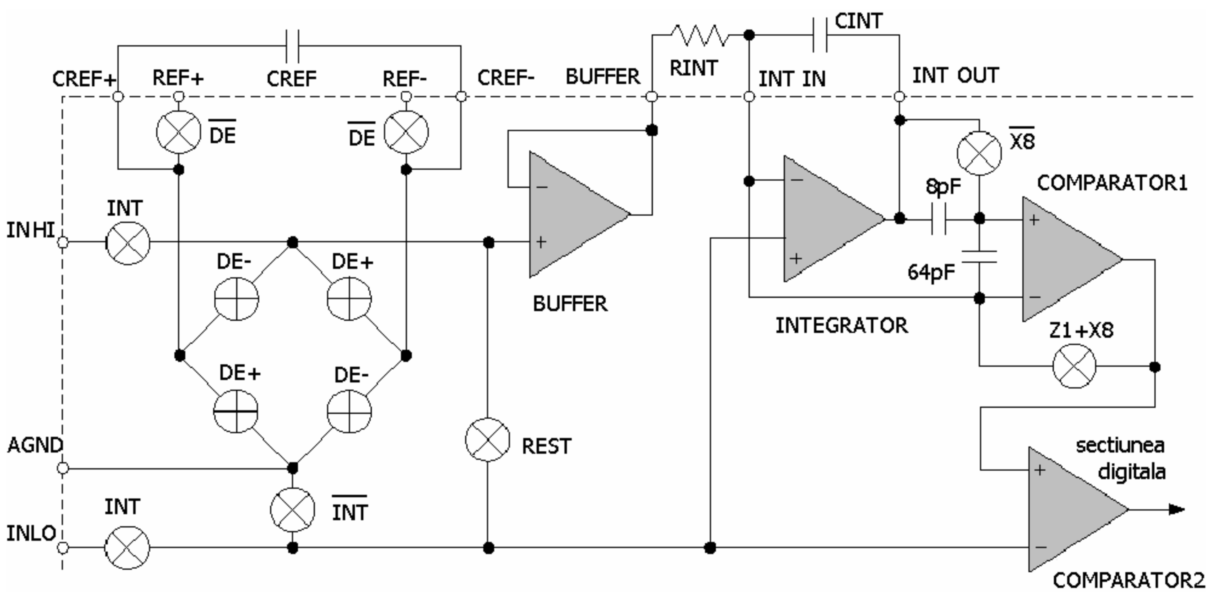


fig.4- 13 Modelul analogic echivalent al circuitelor de intrare în convertorul MAX132

Cum funcţionează comunicaţia cu microcontrolerul ? Datele seriale pe DIN sunt trimise în pachete de 8 biţi şi sunt rotite în registrul intern de 8 biţi ai convertorului pe fiecare front crescător al impulsului de tact CLK. Apoi datele sunt memorate pe frontul crescător al CS fie în registrul de comandă 0 fie în registrul de comandă 1, după cum LSB al octetului de date trimis a fost 0 sau 1 (fig. 4-15). Datele sunt transmise din registrul selectat pe fiecare front căzător al CLK. MSB (D7) este primul bit care trebuie rotit în registru la recepţie, respectiv primul bit care se transmite. Rezultatul conversiei este transmis la ieşire în acelaşi timp cu datele de comandă recepţionate la intrare.

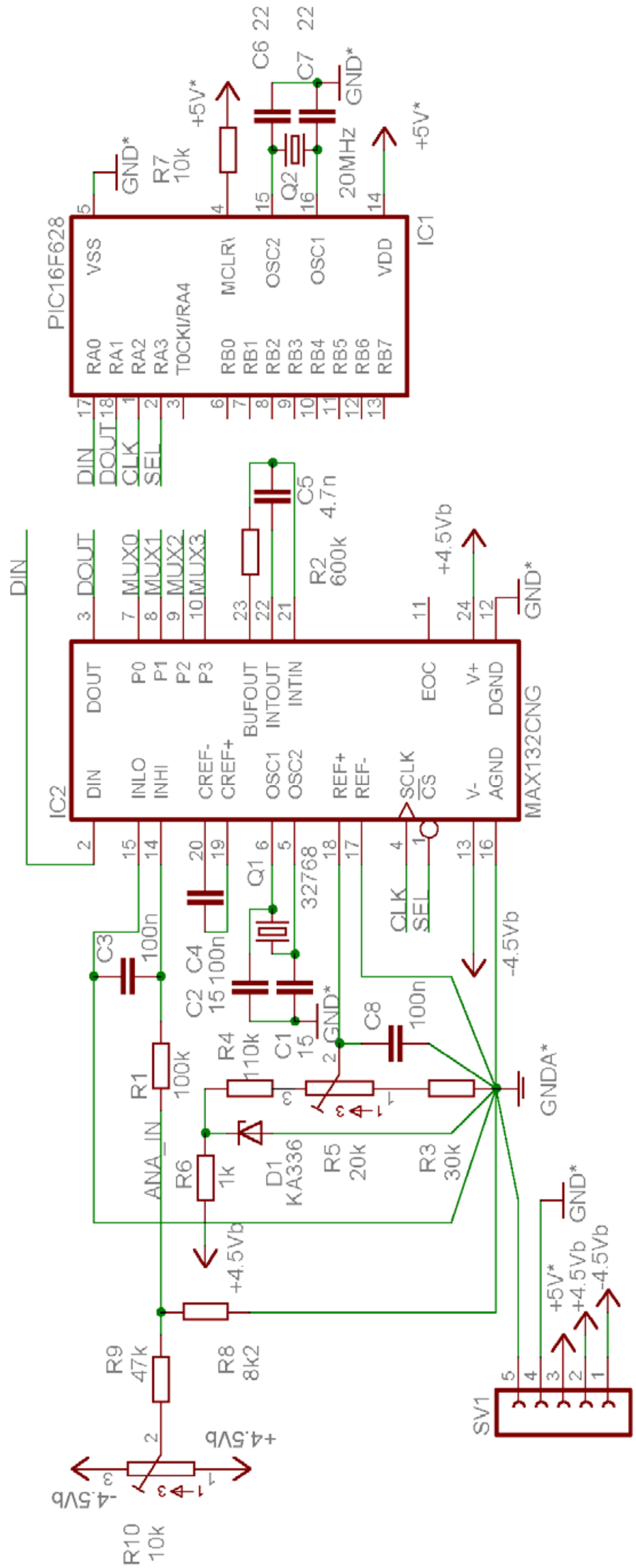


fig.4- 14 Interfațarea unui convertor “meseriaş” la microcontrolerul PIC16F628

<table>
<tr><th colspan="2">REGISTRU</th><th colspan="8">DATA</th></tr>
<tr><th colspan="2"></th><th>D7</th><th>D6</th><th>D5</th><th>D4</th><th>D3</th><th>D2</th><th>D1</th><th>D0</th></tr>
<tr><td rowspan="2">Registrul 0,
intrare comenzi</td><td rowspan="2">1</td><td>Start
conversie</td><td>50Hz</td><td>sleep</td><td>Citeşte
zero</td><td>x</td><td rowspan="2">RS0</td><td rowspan="2">RS1</td><td rowspan="2">0</td></tr>
<tr><td>Revino la
0 la EOC</td><td>60Hz</td><td>treaz</td><td>Citeşte
Vin</td><td>x</td></tr>
<tr><td>Registrul 1,
intrare comenzi</td><td>0</td><td>Setează
P3 ieşire</td><td>Setează
P2 ieşire</td><td>Setează
P1 ieşire</td><td>Setează
P0 ieşire</td><td>x</td><td>x</td><td>x</td><td>1</td></tr>
<tr><td>Registrul 0, ieşire
RS1=0, RS0=0</td><td></td><td>B10</td><td>B9</td><td>B8</td><td>B7</td><td>B6</td><td>B5</td><td>B4</td><td>B3</td></tr>
<tr><td>Registrul 1, ieşire
RS1=0, RS0=1</td><td>1</td><td>B18
MSB</td><td>B17</td><td>B16</td><td>B15</td><td>B14</td><td>B13</td><td>B12</td><td>B11</td></tr>
<tr><td rowspan="2">Registrul status,
ieşire
RS1=0, RS0=0</td><td rowspan="2">0</td><td>coliziune</td><td>EOC</td><td>Integrare
a intrării</td><td>sleep</td><td>Pol +</td><td rowspan="2">B2</td><td rowspan="2">B1</td><td rowspan="2">B0
LSB</td></tr>
<tr><td>Fără
coliziune</td><td>conversie</td><td>Fără
integrare</td><td>treaz</td><td>Pol -</td></tr>
</table>

fig.4- 15 Semnificaţia regiştrilor interni ai convertorului MAX132

Semnificaţia RS0 respectiv RS1 fiind:

RS1	RS0	EFECT
0	0	Selectează registrul0, ieşire pentru B3-B10
0	1	Selectează registrul1, ieşire pentru B11-B18
1	0	Selectează registrul2, ieşire status pentru B0-B2, polaritate, sleep, integrare, EOC, bit de coliziune
1	1	Dată invalidă

fig.4- 16 Biţii de selecţie ai regiştrilor de date

Secvenţa pe care programul implementat de autor o realizează, este definită de tabelul următor:

Registru de instructiune		Data de ieşire
Ciclul 1: start, citeşte status	->	Ieşire în registrul status: EOC, polaritate, B2-B0
Ciclul 2: citeşte MSB	->	Registrul 1: B11-B18
Ciclul 3: citeşte LSB	->	Registrul 0: B3-B10

fig.4- 17 Algoritmul de citire a datelor

Pentru înțelegerea mecanismului de funcționare a integrării cu pantă multiplă se consideră cunoscut principiul integrării dublă pantă, utilizat de majoritatea convertoarelor folosite în aparate de măsură digitale fără pretenții. Spre deosebire de acestea, integrarea cu pantă multiplă execută o serie de funcții suplimentare cum ar fi: corecția automată sau prin program a offsetului, corecția tensiunii reziduale ce rămâne pe condensatorul integrator după trecerea prin zero, tensiune ce generează o eroare aditivă. De exemplu, MAX132 execută trei cicluri distincte (DE-2, DE-3, DE-4) în care această tensiune reziduală este inversată, înmulțită cu 8 şi urmată de o integrare a tensiunii de referință cu o polaritate opusă, încât ieşirea integratorului cade spre zero (fig. 4-18). MAX132 necesită măsurarea tensiunii de offset prin setarea bitului "read zero" în stare logică *high* (fig. 4-15). In acest moment comutatoarele INT şi REST (fig.4-14) sunt închise. Se recomandă citirea a 2…4 valori din convertor şi efectuarea unei medii aritmetice. Se preferă o mediere cu 2^n valori pentru simplificarea calculelor matematice.

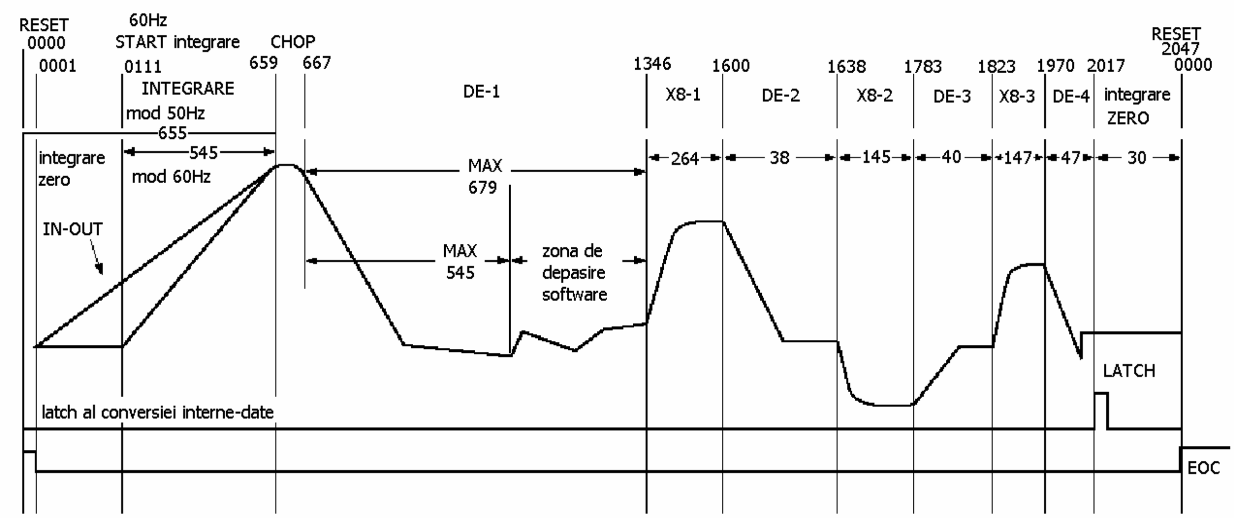


fig.4- 18 Algoritmul de integrare cu pantă multiplă din MAX132

Inceperea conversiei este dictată de bitul D7 (fig.4-15) dacă acesta este setat *high.* Convertorul porneşte imediat conversia, se opreşte la terminarea acesteia şi aşteaptă o nouă comandă de pornire. Pentru obținerea unui flux continuu de date din convertor cea mai bună metodă pare a fi testarea bitului EOC din registrul *Status*. EOC devine *high* la terminarea conversiei. Cum are loc conversia propriuzisă? (fig. 4-18). După măsurarea tensiunii de offset (integrare zero) este integrată tensiunea de referință. Această integrare are o durată fixă prestabilită de 655 impulsuri de tact. La sfârşitul acestei etape (INTEGRARE) urmează faza de integrare cu polaritate opusă (DEintegration-1) dependentă ca durată de momentul trecerii prin zero a semnalului (deci de amplitudinea semnalului de intrare) când comparatorul de ieşire inhibă numărarea impulsurilor. Până în acest moment funcționarea este identică cu a convertorului dublă-pantă. Numărătorul ce acumulează rezultatul este bidirecțional (up-down), sensul de numărare (incrementare sau decrementare) fiind generat de polaritatea integrării în fazele DE-1 pânâ la DE-4. La sfârşitul etapei de integrare a tensiunii de intrare (mijlocul fazei DE-1), numărătorul va conține maxim 512 tacți numărați. Deoarece are loc o multiplicare a tensiunii reziduale conținute de condensatorul de integrare cu 8, numărătorul se va incrementa sau decrementa cu 64 pe parcursul DE-2, cu 8 pe parcursul DE-3 sau cu 1 pe parcursul DE-4, din valoarea inițială memorată înaintea fiecărei faze. Rezultatul acestui numărător este transferat în registrul de rezultat la fiecare terminare

a conversiei. După fiecare fază DE-x (*deintegration*) urmează o fază de pauză care începe la trecerea prin zero a ieşirii integratorului şi durează până la atingerea numărului de impulsuri numărate pentru faza respectivă DE-x. Când trecerea prin zero este detectată la sfârşitul DE-1, integrarea continuă până la următorul tact. Aceasta cauzează o uşoară încărcare suplimentară a condensatorului de integrare. Prima fază de înmulţire cu 8 (X8-1) înversează polaritatea acestei tensiuni şi o înmulţeşte cu 8. DE-2 integrează această tensiune, şi deoarece aceasta a fost înmulţită cu 8, fiecare tact al acestui ciclu este egal cu 1/8 din tactul utilizat în DE-1. La sfârşitul lui DE-2 şi DE-3 are loc X8-2 respectiv X8-3 adică din nou o multiplicare cu 8 a tensiunii reziduale din ciclul precedent, având ca rezultat pentru fiecare fază, o creştere a rezoluţiei de 8 ori faţă de faza precedentă. Terminarea conversiei se face cu o integrare a zeroului, fază care are loc şi la începutul conversiei.

Pentru a economisi pini de microcontroler în cazul utilizării unui multiplexor extern, convertorul are patru pini cu destinaţie selectabilă de către utilizator (P0…P3) selectabili din biţii D4 …D7 ai *Registrului 1 - intrare*. Pe aceştia poate fi interfaţat un multiplexor analogic cu până la 16 canale. *Registrul 0 - ieşire* conţine biţii B3-B10 ai rezultatului conversiei. *Registrul 1 - ieşire* conţine biţii B11-B18 ai rezultatului conversiei în complement faţă de 2 (bitul de polaritate, msb din octet este 1 pentru polarităţi negative). *Registrul Status* conţine biţii B0-B2 care trebuiesc citiţi utilizând o mediere aritmetică pentru a creşte precizia. Bitul de polaritate (D3) va indica dacă citirea este sau nu în depăşire. Bitul de semnalizare a fazei de integrare (D5) este *high* la începutul fazei de integrare şi devine *low* la sfârşitul acesteia. Indică momentul când se poate modifica tensiunea la intrarea convertorului fără a afecta rezultatul conversiei curente. Bitul de coliziune (D7) avertizează microcontrolerul că regiştrii de date au fost modificaţi pe parcursul citirii acestora. Determinarea corectă a stării de coliziune se face înaintea şi după citirea regiştrilor de ieşire *Registrul1* şi *Registrul 2.*

Programul exemplificat execută doar procedura de serializare şi comunicaţie cu convertorul:

```
include f628_20
include jpic628
include max132p
include hd447804
include jprint

procedure init is
 clk = low
 cs = high
end procedure

procedure write_command ( byte in wordd ) is
asm bcf cs
 for 8 loop
  assembler
    bcf   clk       ; clk low
    bcf   dout      ; dout low
    btfsc wordd, 7  ; testează dacă nu e low
    bsf   dout      ; dout = word, x  x = 0...7
    bsf   clk       ; front crescător
    nop             ; min 400nS
    rlf   wordd, f  ; roteşte la stânga
    clrc            ; curăţă carry
```

```
  end assembler
 end loop
asm bsf cs
end procedure

procedure read ( byte in wordd, byte out result ) is
asm bcf cs
result = 0
 for 8 loop
  assembler
    bcf   clk       ; clk low
    bcf   dout      ; data_out low
    btfsc wordd, 7  ; bitul msb al octetului wordd este low ?
    bsf   dout      ; nu, data_out = word, x
    bsf   clk       ; clk front crescător, 200nS
    rlf   wordd, f  ; da, roteşte spre stânga, msb primul,200nS
    clrc            ; curăță carry, 200nS
    bcf   clk       ; clk front căzător
    btfsc din       ; data_in este zero ?
    bsf   status_c  ; nu, setează flagul carry
    rlf   result, f ; da, roteşte, msb primul
    clrc            ; curăță carry pentru operația următoare
  end assembler
 end loop
asm bsf cs
end procedure

var byte mstatus
var byte reg0
var byte reg1
var byte command

init
hd44780_clear

forever loop

; read ( 0b_1101_0100, mstatus )    ; start, 50Hz, awake, read zero, read status
; secvența de mai sus este necesară pentru citirea offsetului
 read ( 0b_1100_0100, mstatus )     ; start, 50Hz, awake, read vin, read status, b3-b0
; write_command ( 0b_1010_0001 )    ; setează P3 on, P2 off, P1 on, P0 off

var bit colision at mstatus   : 7
var bit eoc      at mstatus   : 6
var bit polarity at mstatus   : 3

 delay_20mS ( 10 )

; read ( 0b_0101_0010, reg1 )     ; read zero, b18-b11
; read ( 0b_0101_0000, reg0 )     ; read zero, b10-b3
;  secvența anterioară este necesară pentru calibrarea ofsetului

 read ( 0b_0100_0000, reg0 )       ; read vin, b18-b11
 read ( 0b_0100_0010, reg1 )       ; read vin, b10-b3
```

```
 hd44780_line1
 print_binary_8 ( hd44780, reg0, "0" )
 hd44780_position1 ( 8 )
 print_binary_8 ( hd44780, reg1, "0" )
 hd44780_line2
; print_binary_8 ( hd44780, command, "0" )
; print_binary_8 ( hd44780, mstatus, "0" )

end loop
```

Precauțiile care trebuiesc luate la transpunerea în realitatea abjectă ce ne înconjoară, respectiv la realizarea pe cablaj imprimat a circuitului, sunt numeroase: plan de masă pentru masa analogică, conexiunea într-un singur punct a masei digitale şi a celei analogice, alimentarea secțiunii analogice din baterii care să asigure ±4.5Vb pentru obținerea rezoluției maxime a convertorului, utilizarea unei referințe de tensiune precise şi stabile (KA336 asigură o tensiune de +2.5V, stabilitate 20ppm, care trebuie divizată de grupul R3, R4, R5). Şi o ultimă precauție: acest convertor este extrem de costisitor dacă se procură de la *dealer*ii de componente electronice din țară…

4.8 Convertorul AD de 14 biti MAX121, exemplu de interfațare

MAX121 este un convertor de 14 biți ce face parte din categoria *sample & hold* cu aproximații succesive. Cu o ieşire serială de tipul **S**erial **P**heripheral **I**nterface, o viteză de conversie garantată de 308ksps (**k**ilo **s**amples **p**er **s**econd) adică un timp de conversie de 2.9µS şi un domeniu de variație permis pentru mărimea analogică de intrare (analog **F**ull **S**cale) de ±5V, este un convertor uşor de interfațat fie cu microcontrolere având interfață SPI hardware (se poate obține viteza maximă de conversie), fie cu microcontrolere ce simulează prin software interfața SPI (viteza comunicației este dependentă de frecvența oscilatorului acestuia şi abilitățile de programator ale utilizatorului). Din fig.4-19 se poate observa alimentarea convertorului la +5V şi –12V sau –15V. Puterea totală consumată este în jur de 210mW. Flotarea convertorului față de microcontroler este deasemenea posibilă (nu este prezentată în schemă pentru creşterea inteligibilității) utilizând doar patru optocuplori de viteză (datele sunt bidirecționale) cu intrare trigger schmitd sau optocuplori clasici de viteză cu fotodiodă şi fototranzistor pentru cele trei semnale de comandă: CLKIN, CONVST şi SDATA, plus un pin suplimentar de comandă a direcției pentru SDATA. Structura echivalentă a intrării analogice a convertorului din fig.4-20, ilustrează cum arată aproximativ circuitul de eşantionare- memorare. Condensatorul de memorare este încărcat prin intermediul unui *buffer*, pentru a scădea cât mai mult timpul de achiziție între două conversii. Impedanța echivalentă a intrării apare ca 6K în paralel cu capacitatea parazită a capsulei de 10pF. Intre conversii, intrarea bufferului este conectată cu intrarea analogică prin rezistența de intrare. In momentul începerii conversiei, bufferul este deconectat de la intrarea analogică, iar la sfârşitul acesteia este reconectat la intrare, în timp ce condensatorul de memorare menține valoarea tensiunii de încărcare, egală cu cea de intrare. Comutatorul *track & hold* este pe poziția *track* tot timpul când nu are loc o conversie. Modul *hold* se

inițiază la aproximativ 10nS după inițierea unei conversii. Referința internă de –5V alimentează convertorul DA intern. Ieşirea referinței la pinul Vref trebuie filtrată la masă

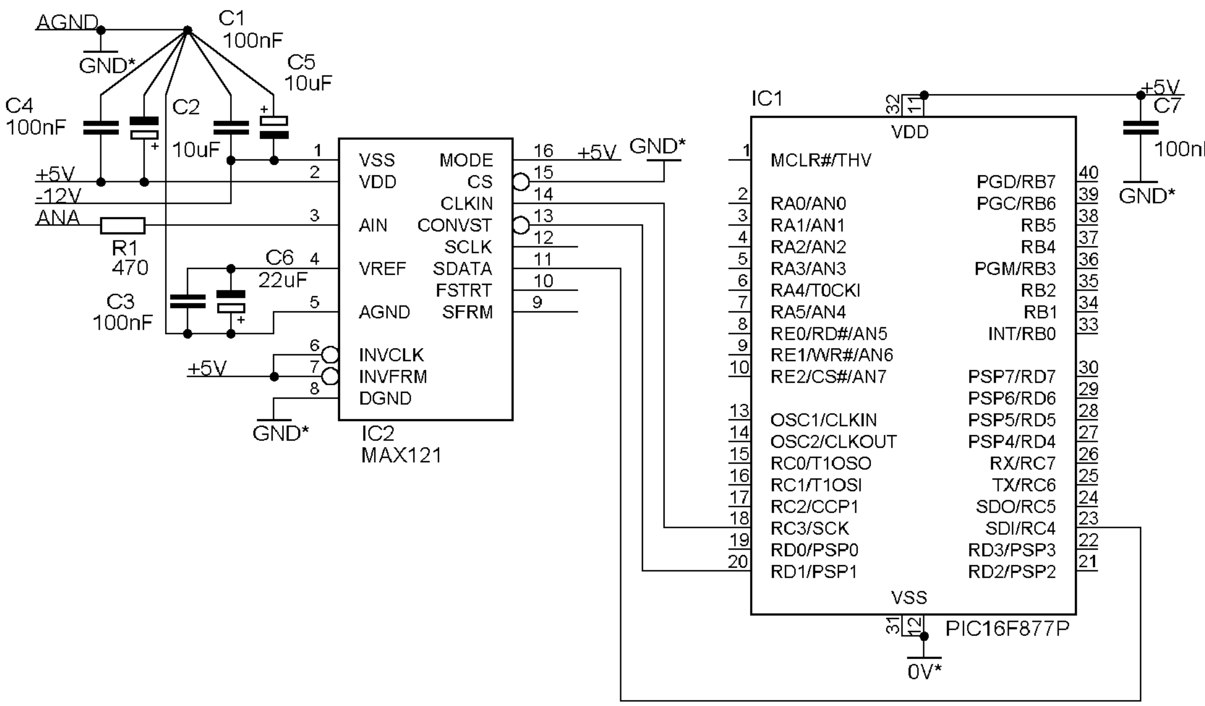


fig.4- 19 Interfațarea convertorului MAX121 prin hardware SPI cu PIC16F87x

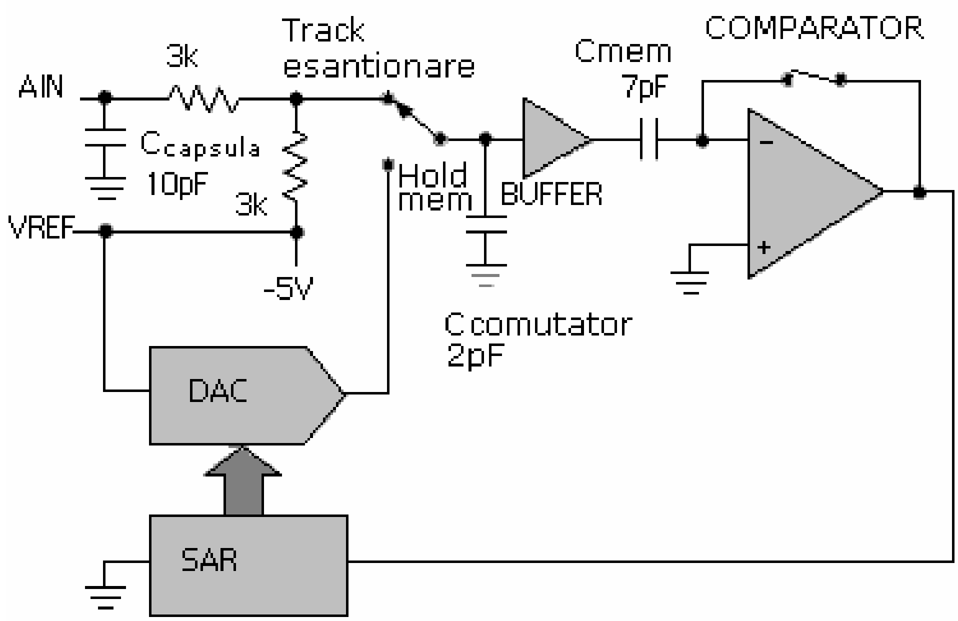


fig.4- 20 Modelul analogic simplificat al convertorului MAX121

cu un grup de condensatori (polarizat şi ceramic). Este posibilă şi utilizarea unei referinţe externe de tensiune, dar aceasta trebuie să aibă valoarea în limitele -5.05V…-5.10V.

Tactul necesar pentru MAX121 trebuie să fie compatibil TTL/CMOS şi să fie în domeniul 100kHz…5.5MHz. Rezultatul conversiei este un şir serial (*data stream)* de 16 biţi de date, pornind cu MSB (bitul14) iar după LSB urmează doi biţi 0 (în total 16 biţi). Formatul datelor de ieşire este în complement faţă de 2, fiecare bit de date este transmis pe pinul SDATA pe frontul crescător al CLKIN. Sunt posibile trei moduri de operare a convertorului:

1. /CONVST este utilizat pentru controlul începerii conversiei. Se foloseşte pentru ***D****igital* ***S****ignal* ***P****rocessing* (prelucrare digitală a semnalelor) când intrarea analogică trebuie citită la intervale foarte precise de timp

2. /CS controlează începerea conversiei. Se foloseşte în situaţia când mai multe dispozitive seriale împart aceeaşi magistrală de date. Când /CS este *high* ieşirile convertorului se găsesc în stare de impedanţă ridicată

3. modul de conversie continuă, destinat achiziţiei de date de tip data-logger, unde convertorul este conectat la microcontroler prin intermediul unei memorii FIFO (**f**irst **i**n **f**irst **o**ut).

Aplicaţia prezentată realizează operarea conversiei în modul 1. Un front negativ pe /CONVST trece comutatorul Track/Hold pe poziţia H şi porneşte conversia în registrul de aproximaţii succesive SAR (fig.4-20). FSTRT care este în mod normal *low* (fig.4-19), trece în *high* pe următorul tact al CLKIN şi rămâne aşa pe durata a unui tact. Pe următorul front crescător al CLKIN, FSTRT devine *low,* şi SFRM trece în stare logică *low,* (dacă /INVFRM este conectat la Vdd). SFRM indică faptul că MSB este gata de a fi memorat. SFRM rămâne low pentru următorii 16 tacţi (14 date + 2 zerouri). Comutatorul T/H revine în poziţia T după ce ultimul bit de date (D0) este transferat pe pinul SDATA. Iniţierea unei noi conversii se poate face după trecerea timpului de achiziţie de 400nS, printr-o nouă comandă pe pinul /CONVST. Pentru a putea iniţia conversia în modul 1, /CS trebuie să fie *low*.

Sunt necesare câteva precizări privind comunicaţia SPI. Standardul SPI necesită ca transferul blocului de date să aibă loc la nivel de octet, dar ieşirea MAX121 transferă date la nivel de 16 biţi. De aceea sunt necesare două operaţii de citire succesivă a câte unui octet, pentru a recepţiona întregul pachet de date de 14 biţi de la convertor. Conversia este iniţiată prin trecerea pinului de comandă IO al microcontrolerului în stare *low*. Apoi este necesară o operaţie de scriere dinspre PIC, chiar dacă aceasta nu conţine o informaţie reală (*dummy data*) pentru a activa tactul serial şi a citi primii 8 biţi de date de la MAX121. Tranziţia datelor la ieşirea convertorului se face pe frontul pozitiv al tactului în timp ce procesorul citeşte datele pe frontul căzător al tactului (pentru CKP = 0).

```
-- file      : max121p.jal
-- date      : 28.01.2002
-- purpose   : hd44780 configuraţie de bază, pini MAX121
-- used by   : hd447804

var volatile bit  hd44780_4_DI is pin_d2
var volatile bit  hd44780_4_E  is pin_d3
var volatile byte hd44780_4_D  is port_d_high

procedure _hd44780_4_init is
   port_d_high              = 0
   pin_d2                   = low
   pin_d3                   = low
```

```
   port_d_high_direction       = all_output
   pin_d2_direction            = output
   pin_d3_direction            = output
end procedure

var volatile bit convst is pin_d1
pin_d1_direction = output
var volatile bit clk is pin_c3
pin_c3_direction = output
var volatile bit sdata is pin_c4
pin_c4_direction = input

-- max121 CLKIN (pin 14) conectat la 877 c3 ( pin 18-SCLK )
-- max121 CONVST(pin 13) conectat la 877 d1 ( pin 20-uz general )
-- max121 SDATA (pin 11) conectat la 877 c4 ( pin 23-SDIN )
```

Programul de citire al datelor din convertor utilizând biblioteca de configurare a pinilor, MAX121p este prezentat aici:

```
include f877_20
include max121p
include hd447804
include jprint
var byte msb, lsb

-- iniţializare SPI, modul master
-- -----------------------------
convst = high ; stop conversie, master mode
-- sspsat: smp_cke_p_s_r/w_ua_bf
--                               0  sspbuf = gol
--                               1  sspbuf = plin, recepţia completă
--                            x
--                        x
--                     x
--                   x
--               0  CKP = 0, data transmisă pe frontul căzător al clk
--           0  data de intrare eşantionată la mijlocul duratei de ieşire a datei
bank_1
f877_sspstat = 0b_0000_0000
bank_0
-- sspcon: wcol_sspov_sspen_ckp_sspm3_sspm2_sspm1_sspm0
--                                 0      0      0      0   spi master
--                             0 = clock inactiv pe nivel low
--                       0 dezactivează portul serial
--                       1 activează portul serial şi configurează sck,sdo,sdi,ss
--                1 = a fost recepţionat un octet , sspbuf memorează data anterioară
--                0 = nu a avut loc depăşirea
--          x

f877_sspcon = 0b_0010_0000 ; fosc/4
hd44780_clear
```

forever loop

```
convst = low                ; porneşte conversia
f877_sspbuf = 0             ; trimite orice caracter (dummy )
bank_1
while ! bf loop end loop    ; aşteaptă până la recepţia primului caracter
bank_0
msb = f877_sspbuf
f877_sspbuf = 0             ; trimite din nou orice caracter pentru generarea clk
bank_1
while ! bf loop end loop
bank_0
lsb = f877_sspbuf
convst = high

hd44780_line1
 print_binary_8 ( hd44780, msb, "0" )
hd44780_position1 ( 8 )
 print_binary_8 ( hd44780, lsb, "0" )
delay_100mS ( 3 )
end loop
```

Modulul SPI (**S***ynchronous* **P***heripheral* **I***nterface*) echipează doar microcontrolerele flash serioase ca 16F87x, 16F7x, sau mai tânărul 16F87xA, însă implementarea acestui protocol poate fi făcută şi cu modulul USART funcţionând în modul sincron, disponibil şi în seria 16F62x. Inţelegerea funcţionării interfeţei SPI este destul de greoaie datorită multitudinii de posibilităţi de configurare a acesteia. După cum exemplul de mai sus o evidenţiază, modulul operează cu trei semnale: **S***erial* **C***lo***K**, **S***erial* **D***ata* **IN***put* şi **S***erial* **D***ata* **OUT***put*. Ideea de bază este că SDIN şi SDOUT funcţionează sincron cu SCK, atât octetul de intrare cât şi cel de ieşire sunt generate simultan. Se poate utiliza şi un al patrulea pin de comandă **S***lave* **S***elect* numai pentru situaţia când microcontrolerul este în mod sclav, ceea ce nu este cazul în situaţia de faţă. Iniţializarea modulului SPI implică setarea unor biţi din regiştrii SSPCON şi SSPSTAT după cum se observă în fig.4-21 respectiv fig.4-22 şi în exemplul jal prezentat. Pentru simplificarea lucrurilor, fig.4-21 şi fig4-22 tratează doar funcţia biţilor utilizaţi în modul SPI (aici R = read, W = write).

Cei mai importanţi biţi ai registrului SSPSTAT (fig.4-22) şi SSPCON (fig.4-21) referitori la parametrii comunicaţiei, sunt biţii CKE, SMP şi CKP. Dacă configurarea acestora nu este făcută în concordanţă cu semnalul ce urmează a fi recepţionat din convertor este posibil să nu recepţionăm niciodată un şir de date valid, mai ales că există un număr mare de combinaţii ce poate fi făcut cu valorile logice ale acestor regiştrii. Din fig.4-22 şi fig.4-23 se observă flexibilitatea interfeţei SPI: datele pot fi transmise respectiv recepţionate atât pe frontul pozitiv cât şi pe frontul negativ al tactului, polaritatea acestuia putând fi pozitivă (*active high*) sau negativă (*active low*) după cum combinaţia CKP + CKE o generează. Inclusiv eşantionarea citirii datelor de intrare poate avea loc la mijlocul sau la sfârşitul timpului de generare a datelor de ieşire (după valoarea logică a SMP), facilitate importantă pentru periferice ce necesită un timp de stabilizare a datelor din momentul accesării şi până la momentul citirii.

WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
7 R/W	6 R/W	5 R/W	4 R/W	3 R/W	2 R/W	1 R/W	0 R/W
SSPOV: bit de indicare al depăşirii valorii maxime a registrului la recepție Pentru mod SPI: 1 = un nou octet s-a recepționat cât timp SSPBUF memorează data anterioară. Data din SSPR este pierdută la depăşire. In mod sclav, utilizatorul trebuie să citească SSPBUF, chiar dacă transmite numai date, pentru a preîntâmpina depăşirea. In mod stăpân, acest bit nu este setat, operația fiind inițiată prin scriere în registrul SSPBUF 0 = nu a avut loc depăşirea							
SSPEN: bit de setare al modului Synchronous Serial Port Mod SPI: pinul trebuie configurat corespunzător ca intrare sau ieşire 1 = activează portul serial şi configurează SCK,SDO,SDI şi SS ca pini ai portului serial 0 = dezactivează portul serial şi configurează pinii de mai sus ca pini IO cu utilizare generală							
CKP: bitul de selecție al polarității CLK Pentru mod SPI: 1 = starea inactivă a CLK este high 0 = starea inactivă a CLK este low							
SSPM3:SSPM0: biții de selecție ai portului sincron serial 0000 = SPI, stăpân, clock = fosc/4 0001 = SPI, stăpân, clock = fosc/16 0010 = SPI, stăpân, clock = fosc/64 0011 = SPI, stăpân, clock = TMR2 output/2 0100 = SPI, sclav, clock = pinul SCK , SS este activat 0101 = SPI, sclav, clock = pinul SCK, controlul via SS dezactivat, SS poate fi folosit ca pin IO SSPCON							

fig.4- 21 Registrul SSPCON destinat setărilor pentru comunicația SPI

SMP	CKE	D/A	P	S	R/W	UA	BF
7 R/W	6 R/W	5 R	4 R/W	3 R	2 R	1 R	0 R
SMP: bit de eşantionare Pentru mod SPI stăpân: 1 = data de intrare este eşantionată la sfârşitul duratei datei de ieşire 0 = data de intrare este eşantionată la mijlocul duratei datei de ieşire pentru mod SPI sclav: SMP trebuie resetat în acest mod							
CKE: flag pentru selecția frontului tactului SPI Mod SPI: dacă CKP = 0, 1 = data se transmite pe frontul crescător al SCK 0 = data se transmite pe frontul descrescător al SCK dacă CKP = 1, 1 = data se transmite pe frontul descrescător al SCK 0 = data se transmite pe frontul crescător al SCK							
BF: bit de status ce semnalizează umplerea bufferului (buffer full) La recepție (mod SPI şi I2C): 1 = recepția commpletă, SSPBUF este plin 0 = receptia incompletă, SSPBUF este gol SSPSTAT							

fig.4- 22 Biții corespunzători setărilor pentru SPI în registrul SSPSTAT

O explicație ceva mai elegantă este oferită de diagrama de timp a comunicației SPI pentru modul master al microcontrolerului:

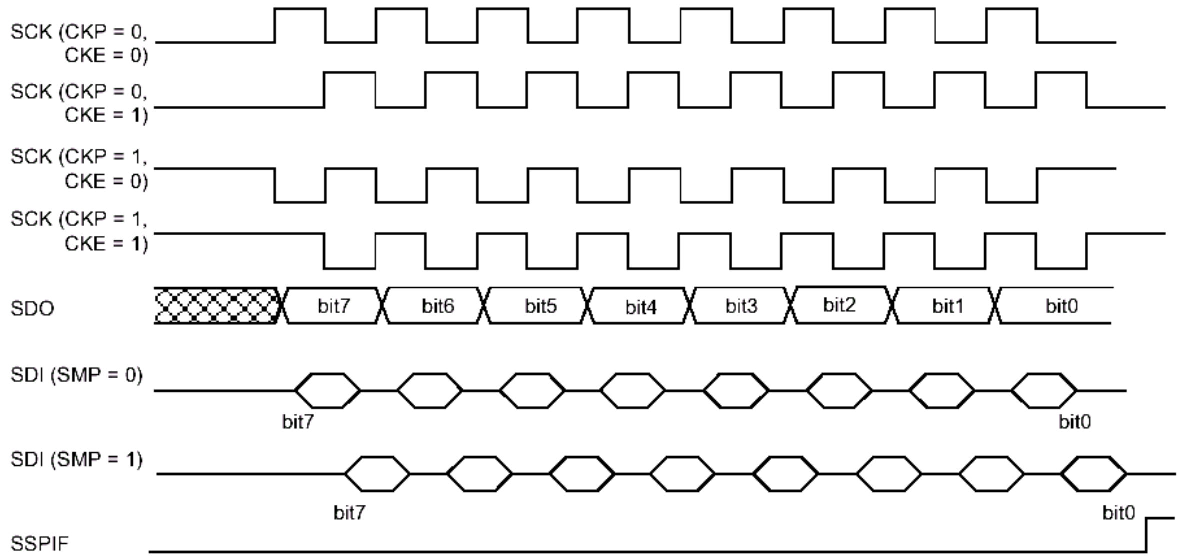


fig.4- 23 Diagrama de timp a modului SPI (PIC16F87x) pentru modul master

Dacă se intenționează conectarea a două microcontrolere în modul SPI, unul dintre acestea poate fi master iar celălalt slave. Modul de setare al modului slave se găseşte în [14] sau: CD:\datsheet\microchip\pic16f87xc în capitolul "Master Synchronous Serial Port module".

4.9 Convertorul AD dual de 12 biți, MCP3202

Dacă ați analizat cu atenție posibilitățile convertoarelor AD ale seriei Microchip midrange, atunci ați observat cu siguranță că producătorul nu a reuşit să treacă de granița celor 10 biți, reprezentând rezoluția maximă a convertorului AD intern microcontrolerului. Cele mai performante convertoare AD independente produse de Microchip, nu depăşesc limita celor 12 biți, (la momentul editării acestui material) şi MCP3202 este unul dintre acestea. Integratul face parte din familia convertoarelor cu aproximații succesive având deasemenea *sample&hold* (eşantionare-memorare). Cele două intrări analogice pot fi utilizate separat sau ca o singură intrare pseudo-diferențială. O particularitate neplăcută este lipsa intrării separate pentru tensiunea de referință, aceasta fiind generată din tensiunea de alimentare, stabilitatea şi valoarea acesteia afectând conversia. Protocolul de interfațare al convertorului este acelaşi SPI prezentat în capitolul anterior cu trei semnale de comandă: CLK, DIN şi DOUT. Intrarea CLK este utilizată pentru a iniția conversia şi pentru a sincroniza datele introduse sau extrase din convertor. Intrarea DIN se utilizează pentru transferul datelor de configurare a canalelor analogice în regiştrii interni convertorului. Ieşirea DOUT este destinată extragerii rezultatului conversiei. Suplimentar există şi un pin de comandă /CS/SHDN, acesta este utilizat pentru inițierea comunicației cu convertorul (când este în stare logică *low*) şi încheierea conversiei concomitent cu trecerea dispozitivului în stare de consum redus (când este în stare logică *high*). Intre două conversii /CS/SHD trebuie să fie menținut în stare logică *high*.

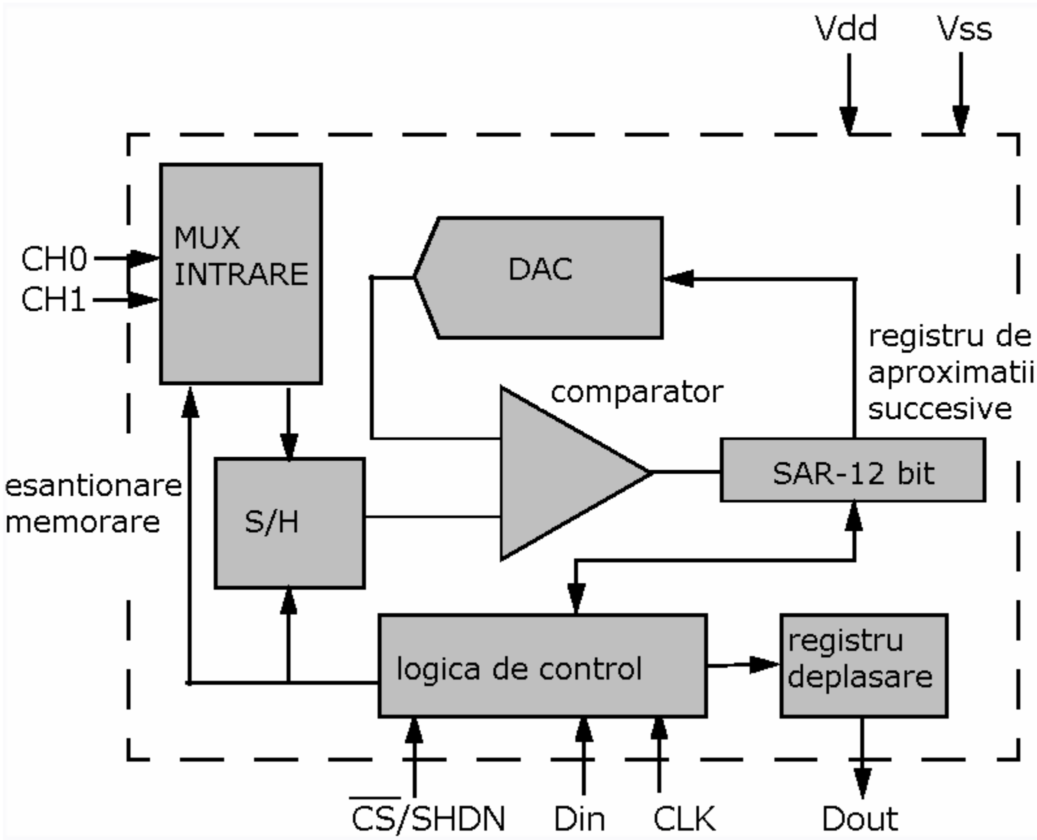


fig.4- 24 Schema bloc a convertorului MCP3202

Cum arhitectura prezentată în fig.4-24 vă este deja familiară din capitolul anterior şi explicarea funcţionării acesteia va fi mult mai inteligibilă. Un eşantion din tensiunea de intrare este achiziţionată şi memorată pe condensatorul de memorare, un timp de 1.5 durate de tact, începând cu al doilea front crescător al tactului după ce bitul de start a fost recepţionat. După încheierea acestei perioade de timp, comutatorul de eşantionare memorare se deschide şi tensiunea memorată pe condensator este utilizată pentru producerea codului digital de 12 biţi. Rata maximă de conversie este de 100ksps. Modul de configurare analogică a intrării este dependent de biţii de configurare care se transmit serial, imediat după bitul de startare a conversiei:

	Biţi de configurare		**Selecţia canalului**		GND
	SGL/DIFF	ODD/SIGN	0	1	
Mod unipolar	1	0	+		-
	1	1		+	-
Mod pseudo diferenţial	0	0	In+	In-	
	0	1	In-	In+	

fig.4- 25 Biţii de configurare ai MCP3202

Schema analogică echivalentă a convertorului este indentică cu cea din convertorul AD intern al PIC16F87x, prezentată în fig.4-12, cu deosebiri minore privind capacitatea de memorare care pare a fi doar 20pF, şi o rezistență internă a comutatorului de eşantionare memorare de 1Kohm, care obligă utilizatorul la utilizarea unei surse de semnal cu impedanță de ieşire cât mai aproape de 0. Acest impediment se datorează constantei de încărcare al condensatorului de eşantionare:

T = (Rs + Rss) x Csample unde:
Rs este impedanța de ieşire a sursei de semnal
Rss este impedanța comutatorului de eşantionare
Csample este capacitatea condensatorului de eşantionare

Inafara acestei limitări ce apare la toate convertoarele microcontrolelor Microchip, o altă limitare importantă este excursia tensiunii analogice de intrare pentru modul pseudo-diferențial, şi anume excursia Vin+ este în domeniul [Vin- …Vref(Vref + Vin-)] în timp ce (Vin-) = ±100mV, măsurat față de masă. Dacă (Vin+) <= (Vin-), codul rezultat va fi 000h. Dacă (Vin+) >= (Vref + Vin-) – 1LSB, codul rezultat va fi FFFh. Ecuația ce guvernează conversia, pentru situațiile ce nu se încadrează în restricțiile de mai sus este:

Digital Output Code = 4096*Vin/VCC unde: Vin este tensiunea de intrare
Vcc este tensiunea de alimentare

Există două moduri de a prelua informația digitală din convertor:
- ♦ Formatul cu MSB transmis primul
- ♦ Formatul cu LSB transmis primul

Vom analiza doar primul format, fiind cel mai inteligibil. In fig4.26 se observă secvența de inițializare: bitul de start este generat pe parcursul primului impuls de tact după ce /CS a trecut în stare *low*. Următorii doi biți selectează modul de lucru al convertorului: două intrări unipolare sau o intrare pseudo-diferențială (SLG/DIFF), respectiv selecția canalului şi polaritatea semnalului pentru modul pseudo-diferențial (ODD/SIGN). Imediat după bitul ODD/SIGN, este transmis spre convertor bitul de selecție al modului de transfer, acesta este *MSB first* pentru MSBF *low* respectiv *LSB first* pentru MSBF *high*, cu precizarea că prima transmisie a datelor dinspre convertor, în acest al doilea caz, va fi tot *MSB first*. Pe frontul căzător al MSBF, convertorul generează un bit nul, următorii 12 tacți secvențiali vor genera rezultatul conversiei. Datele sunt generate întotdeauna pe frontul căzător al impulsului de tact. Este posibil să se mențină /CS în stare logică *low* şi să se genereze un octet de comandă nul (conținînd datele egale cu zero) înaintea bitului de start.

Funcționarea acestui convertor seamănă ca două picături de apă cu exemplele precedente, motiv pentru care lăsăm cititorului plăcerea de a modifica exemplele de program implementate anterior pentru a obține o comunicație validă cu convertorul. O precizare ajutătoarea ar fi următoarea: deoarece DIN şi DOUT nu sunt active simultan (fig.4.26) ele pot fi conectate împreună dacă se optează pentru utilizarea unui protocol SPI generat prin software. Timpul minim de dezactivare prin trecerea CS *high* este min. 0.5µS, întârzierea minimă între CS *low* şi primul front crescător al CLK este de 100nS iar timpul minim de eşantionare este 1.5 tacți CLK. Timpul de conversie dureaza maxim 12 tacți, pe durata celui mai puțin semnificativ bit B0 comparatorul intern este dezactivat, intrarea de referință a acestuia devenind stare de înaltă impedanță în timp ce tactul CLK transferă spre ieşire valoarea bitului B0.

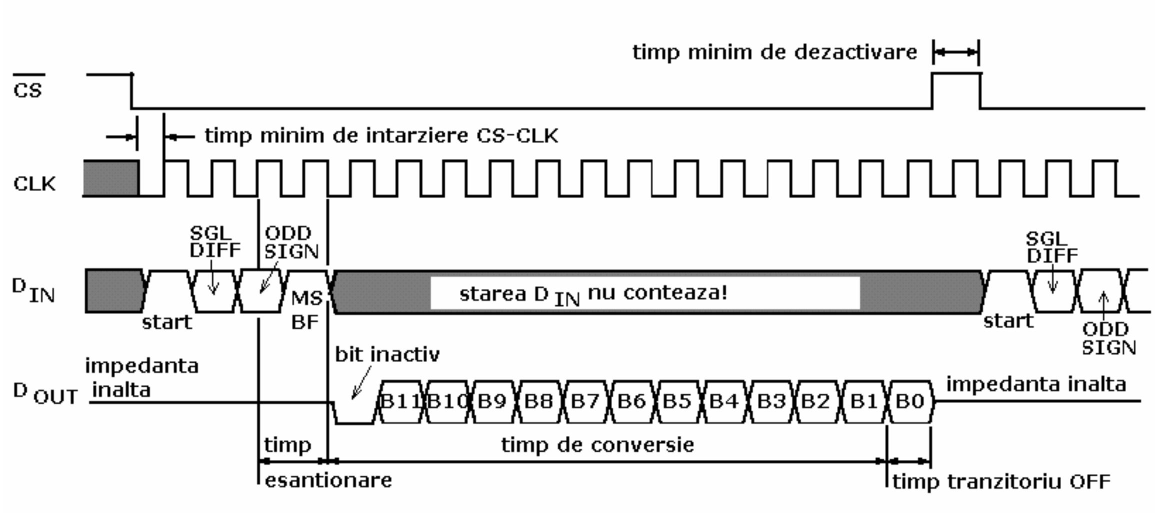
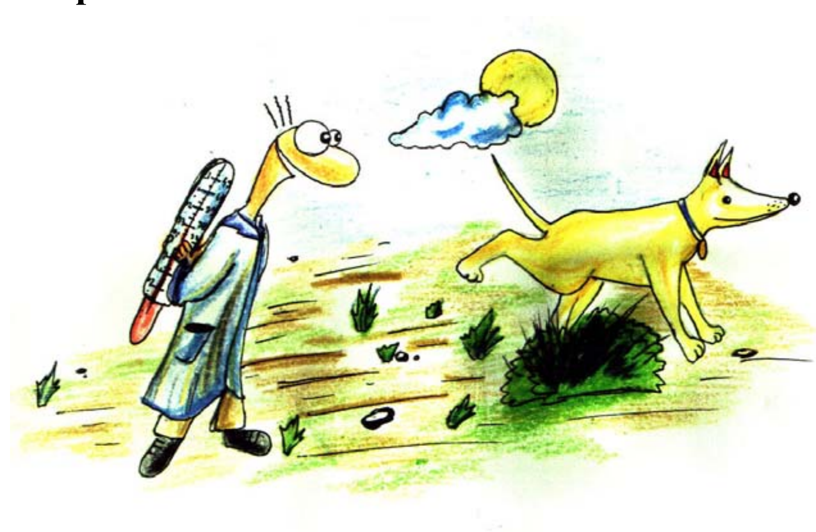


fig.4- 26 Transmisia datelor în formatul *MSB first*

4.10 Măsurarea temperaturii



Se pare că orice om are păsărica lui. Unii au chiar câte un stol întreg. Lipsă. Parcurgând rapid acest capitol se pare că metodele de măsură a temperaturii ambiante fac şi ele parte din din stolul meu. Există cel puţin patru categorii de senzori destinaţi măsurării temperaturii:

- senzori analogici semiconductori:
  - diode
  - tranzistoare

   - circuite integrate specializate (termorezistențe integrate sau “diode” cu pantă îmbunătățită)
- senzori pasivi (necesită polarizare externă) :
   - termistoare PTC şi NTC
   - termorezistențe
- senzori activi (nu necesită polarizare):
   - termocuple
   - senzori indicatori cu bimetal
- Senzori inteligenți cu ieşire digitală pe *bus* cu unu, două sau trei fire

şi două mari clase de metode de măsură:

- măsurarea temperaturii prin contact direct al senzorului cu mediul de măsurat
- măsurarea temperaturii fără contact cu mediul prin metode radiative

Presupunând că dumneata, destinatarul acestor rânduri eşti deja un potențial emigrant pentru Canada care cunoşti cum se măsoară temperatura utilizând metodele descrise mai sus, voi face o trecere în revistă foarte sumară a teoriei de funcționare a acestor dispozitive.

4.10.1 Dispozitive semiconductoare cu joncțiune (diode şi tranzistori)

Orice joncțiune semiconductoare parcursă de un curent constant, va produce o cădere de tensiune la terminalele ei a cărei valoare este dependentă de tehnologia de realizare a joncțiunii şi de temperatura ambiantă la care aceasta funcționează. De exemplu, pentru o diodă semiconductoare de 500mW pe plachetă de siliciu, funcționând la 25C, tensiunea anod-catod va avea valoarea cuprinsă între 0.5V şi 0.7V pentru un curent de polarizare constant. Acest curent de polarizare poate produce încălzirea joncțiunii prin disipație termică, motiv pentru care, dacă joncțiunile sunt utilizate în scopul măsurării temperaturii, vor necesita un curent de polarizare mic (între 100uA şi maxim 1mA). Ecuația care stă la baza măsurării de temperatură (dar şi la efectuarea operatiilor matematice log, antilog şi raport prin metode analogice) este:

$$Id = Is(e^{\frac{Vd}{kVt}} - 1)$$

Unde: ***Id*** este curentul prin joncțiune [A]
Vd este căderea de tensiune pe joncțiune [V]
Is este curentul de saturație [A]
k este un factor de proporționalitate k = 1…2, creşte cu scăderea ***Id***
respectiv:

$$Vt = kT / q = T / 11000$$

Vt este “tensiunea termică” [V] şi are valoarea cuprinsă între 25…35mV la 20C
T este temperatura joncțiunii [K]
k este constanta lui Boltzman
q este sarcina electronului

Această ecuaţie nu arată unui electronist, decât faptul că încălzirea unei diode semiconductoare, produce o variaţie a tensiunii pe joncţiune de -2.0mV…-2.5mV pentru modificarea temperaturii cu fiecare 1 °C, dacă joncţiunea se menţine sub curent constant. Unui fizician însă, îi poate poate creea sentimente demiurgice, determinindu-l să scrie pagini întregi de teorie.

O primă analiză ne arată că această caracteristică de variaţie a "tensiunii termice" cu temperatura ambiantă, necontrolabilă perfect nici în cadrul aceluiaşi lot de dispozitive similare, face ca dioda sau tranzistorul să nu posede capacitatea de înlocuire directă (*direct replacement*) ceea ce îl face de neutilizat în producţia de serie. Din ecuaţia prezentată mai sus rezultă două metode de măsură a temperaturii cu diode:

- Injectarea unui curent constant în joncţiune şi măsurarea căderii de tensiune pe joncţiune
- Alimentarea joncţiunii cu tensiune constantă şi măsurarea variaţiei curentului prin joncţiune

Din punct de vedere practic, prima variantă are avantaje majore prin faptul ca dioda poate fi montată în bucla de reacţie negativă a unui amplificator operaţional în configuraţie inversoare, a cărui intrare se alimentează de la o referinţă de tensiune. Rezultatul va fi un generator flotant de curent constant performant. Dacă considerăm joncţiunea semiconductare ca fiind chiar joncţiunea bază-emitor a unui tranzistor comun, atunci putem descoperi uşor încă cel puţin două moduri de măsurare a temperaturii:

1) Din ecuaţia statică de funcţionare a trazistorului, redusă la cazul practic (α = 0) Ic = βIb + αIe, împreună cu logaritmarea ecuaţiei anterioare rezultă:

$$\frac{Vbe}{kVt} = \ln \frac{Ib}{Is}$$

Executând o manevră inginerească de ascundere a tuturor constantelor de dispozitiv în aceeaşi contantă k1, (inclusiv logaritmul din constanta Is pe care l-am pierdut pe parcurs tocmai fiincă este o constantă) putem scrie fără lacrimi în ochi:

$$Ib = e^{\frac{Vbe}{k1Vt}}$$

sau fără prea multă imaginaţie:

$$Vbe = k1Vt \ln Ib$$

sau în sfârşit:

$$Ic = \beta \times e^{\frac{Vbe}{k1Vt}}$$

Deci Ic = f (T), dacă se păstrează polarizarea tranzistorului sub tensiune constantă, curentul de colector va fi proporţional cu variaţia de temperatură a joncţiunii bază-emitor. Factorul de amplificare este în acest caz elementul de proporţionalitate.

2) Polarizarea tranzistorului în regiunea activă normală şi măsurarea tensiunii Vce asigurând un curent constant de colector respectiv de bază, va da măsura variației de temperatură ambiantă.

Există încă două metode cunoscute dar care sunt derivate din cele prezentate, şi care nu fac altceva decât să introducă "finețuri" algoritmului deja prezentat. In concluzie, avantajul măsurării temperaturii cu diode şi tranzistoare derivă din simplitatea metodei, dezavantajele sunt însă numeroase:

- Nu sunt reproductibile, deci dispozitivele semiconductoare nu pot fi înlocuite direct ci doar printr-o recalibrare a electronicii de măsură
- Pot măsura temperaturi cuprinse doar între –50C…+100C, nefiind perfect liniare pe întregul domeniu
- Depăşirea accidentală a temperaturii de 125C duce la distrugerea ireversibilă a joncțiunii

4.10.2 Circuite integrate destinate măsurării temperaturii, cu ieşire analogică

Dezavantajul joncțiunilor semiconductoare este corectat de circuitele integrate specializate de măsură a temperaturii. Aspectul acestor circuite integrate este identic cu al tranzistoarelor, având împachetarea în diverse capsule de la cele subminiatură până la capsulele standard TO-xx. Spre deosebire de diode a căror pantă de variație cu temperatura a tensiunii termice este variabilă de la dispozitiv la dispozitiv, circuitele integrate au o variație fixă de 10mV/C sau 25mV/C asigurând o tensiune de ieşire proporțională cu diverse sisteme de măsură a temperaturii (Celsius, Kelvin, Fahreinheit), astfel încât tensiunea de ieşire este egală (cu un factor de corecție) cu temperatura măsurată, pentru sistemul de măsură ales. Astfel de circuite integrate sunt: LM35 (cu echivalentul românesc βM135) pentru ieşire în sistem Kelvin, cu prețul de cost foarte redus, interschimbabili pentru o precizie garantată de ±3°, LM34 (Fahrenheit), LM45 (Celsius), LM50 (Celsius) LM335, AD22100KT etc. Şi aceste circuite integrate au limitări, cea mai deranjantă este necesitatea amplificării semnalului analogic utilizând amplificatoare operaționale, pentru compatibilizarea nivelului de semnal cu intrarea convertorului AD al microcontrolerului. O altă limitare importantă este necesitatea utilizării a câte unui pin AD al microcontrolerului pentru fiecare senzor de temperatură interfațat, respectiv imposibilitatea utilizării unor conexiuni directe foarte lungi între senzori şi microcontroler, datorită mixării zgomotelor peste semnalul util. Corectarea problemelor de zgomot se face utilizând convertoare tensiune-curent în imediata apropiere a senzorului, astfel că transmisia semnalului de temperatură analogic se face în curent (mult mai greu de perturbat), iar la nivelul plăcii microcontroler se revine printr-o conversie curent-tensiune (cea mai simplă conversie este banala rezistență, urmată de un filtru cu condensator) la o mărime compatibilă cu intrarea convertorului AD.

Termorezistențele semiconductoare fac parte din noua generație de dispozitive destinate măsurării temperaturilor medii. Ele nu pot fi utilizate mai sus de 300C…350C, necesită un curent de polarizare sub 1mA şi au aspectul clasic al unei diode semiconductoare în capsulă de sticlă (capsula DO-34). Terminalele acestora trebuiesc conectate prin sudură prin puncte sau prin contact mecanic (cu şurub sau cu presare)

deoarece peste 200C sudurile cu fludor se înmoaie. Variația tipică a rezistenței termorezistenței KTY8X pentru intervalul 20C…300C şi It = 0.5…1mA este de la 300 la 1200 ohmi, aproape liniară şi puternic dependentă de curent. Termorezistențele semiconductoare sunt mult mai ieftine decât termorezistențele cu fir de platină sau cupru, având dezavantajul temperaturii maxime de lucru mai reduse şi a proprietăților mecanice destul de slabe, comparativ cu cele clasice. Gabaritul redus le face să fie extrem de utilizate în aplicații de termocontrol unde de multe ori păstrarea unei inerții termice reduse este esențială.

4.10.3 Senzori pasivi pentru măsurarea temperaturii

Din această categorie fac parte termistoarele şi termorezistențele. Sunt senzori pasivi deoarece necesită polarizare fie în curent fie în tensiune. După modul de variație a rezistenței cu temperatura ambiantă există termistoare cu variație negativă cu temperatura, NTC, a căror lege de variație este:

$$R_t = a \times e^{\frac{b}{T}}$$

Unde: R_t este rezistența termistorului

a, b sunt constante de material, orice producător serios va publica aceste constante în fila de catalog a termistorului respectiv

T este temperatura în grade kelvin

şi termistoare cu lege de variație pozitivă cu temperatura, PTC:

$$R_t = a + c \times e^{\frac{b}{T}}$$

Unde: Rt este rezistența termistorului,

a, b, c sunt constante de material

T este temperatura în grade kelvin

După cum se observă în relațiile de mai sus, variația rezistenței ambelor tipuri de termistoare este puternic neliniară cu temperatura. O primă concluzie este că acest tip de senzori sunt dificil de utilizat, necesitând fie liniarizare prin tabele stocate în memoria microcontrolerului, fie prin rezolvarea unor ecuații matematice nu tocmai simple pentru microcontrolerul de 8 biți. Cu toate acestea, există firme care produc termistori de mare precizie care pot fi interschimbabili, au dimensiuni microscopice şi măsoară foarte precis temperaturi cuprinse între –50C şi +120C. In ceea ce priveşte metodele de conectare în circuit a acestor senzori, sunt clasice conectarea în semipunte de măsură (un senzor), conectarea în punte de măsură (unu sau doi senzori), sau alimentarea sub curent constant a unui singur senzor. Toate metodele enumerate necesită o referință de tensiune, eventual un generator de curent constant pentru a minimiza eroarea de măsură datorată variatiei curentului cu temperatura prin senzor (în metoda semipunte) şi un circuit de amplificare a potențialului cules de pe senzor. Scalarea se face direct în memoria microcontrolerului. Se

practică interfaţarea directă cu convertorul AD sau convertorul tensiune-timp din microcontroler prin alegerea corespunzătoare a rezistenţei iniţiale a senzorului şi a curentului de polarizare astfel încât tensiunea generată pe intervalul de temperatură de măsurat să poată fi preluată cu rezoluţia maximă de către microcontroler. Sunt situaţii când scalarea duce la scăderea rezoluţiei de la 10 biţi la 6…7 biţi. Dacă aplicaţia nu necesită decât o comparare grosieră cu o temperatură de referinţă cu valoare întreagă, reducerea rezoluţiei datorată scalării nu prezintă probleme. Dacă se doreşte însă afişarea cât mai precisă a temperaturii măsurate, este nevoie de rezoluţia maximă a convertorului AD.

Termorezistenţele (**R**esistor **T**emperature **D**ependent) sunt cele mai comune dispozitive de măsură a temperaturii utilizate în automatizări pentru domeniul –200C…+600C (RTD cu platină) respectiv +250C (RTD cu cupru). Se prezintă într-o largă varietate de forme şi dimensiuni, de la cele cilindrice (d =5mm, l =50mm) până la cele subminiatură având aspectul unui film subţire (**T**hin **F**ilm RTD). Există mai multe metode de conectare a acestor termorezistenţe în circuit, pentru a minimiza căderea de tensiune pe terminale datorită curentului de polarizare (amintiţi-vă metoda de măsură a rezistenţei aval şi amonte învăţată în liceu), cele mai cunoscute sunt cu două fire (pentru distanţe foarte scurte), cu trei fire (pentru distanţe lungi, este metoda cea mai utilizată), cu patru fire (măsurători speciale de etalonare)

4.10.4 Senzori activi de măsură a temperaturii

Termocuplele au fost catalogate de autor drept senzori activi deoarece generează tensiune electromotoare. Termocuplele funcţionează pe baza efectului Seebeck îmbunătăţit, (după numele celui ce la pus pentru prima dată în evidenţă, în 1821, Thomas Seebeck) şi anume, dacă două fire metalice cu electronegativităţi diferite sunt puse în contact mecanic (sudură), la capetele lor menţinute la temperatura ambiantă se va produce o diferenţă de potenţial proporţională cu diferenţa de temperatură între joncţiunea încălzită şi temperatura ambiantă, tensiune dependentă de tipul de material aflat în contact. O primă concluzie firească este că fiecare termocuplu având două terminale se transformă într-un grup de trei termocuple prin simpla conectare a terminalelor acestuia cu conductoare având alt material decât cel conţinut de termocuplu. Această concluzie nefericită obligă utilizatorul perfecţionist să folosească numai cabluri speciale realizate din perechi de conductoare din materiale identice cu cele din termocuplu, dacă semnalul se transferă la distanţă, sau să folosească convertoare de semnal în imediata apropiere a termocuplelor (mai ales pentru termocuple S, R, B unde cablul costă o avere). Tipurile existente de termocuple, evidenţiază o variaţie extrem de mare a tensiunii seebeck corespunzătoare variaţiei temperaturii cu 1C.

Determinarea termocuplului optim se face în funcţie de ceea ce dorim să măsurăm. Este la mintea cocoşului că nu vom utiliza niciodată un termocuplu pentru a măsura temperatura ambiantă (-30C…+50C) şi nici un termocuplu S pentru domeniul 200C…400C. Un aspect neplăcut al utilizării termocuplului este compensarea temperaturii joncţiunii reci. Temperatura contactelor termocuplului (unde se culege temperatura măsurată) trebuie fie să fie păstrată la o temperatură fixă (de obicei 0C), fie să fie măsurată cu un senzor de temperatură ambiantă şi apoi efectuată corecţia de măsură. Desigur că eroarea generată de lipsa acestei corecţii pentru o temperatură măsurată de 1000C este suficient de mică ca în practică corecţia să poată lipsi. Nu acelaşi lucru se poate spune

pentru o temperatură măsurată mult mai mică (de exemplu 200C) unde eroarea poate ajunge la 10%. Această compensare complică şi tehnologia de "culegere" a tensiunii de pe terminalele reci, care sunt montate în contact termic cu un bloc metalic pentru a avea amândouă aceeaşi temperatură. Senzorul de temperatură ambiantă se montează pe acest bloc metalic. Se utilizează conectori standardizati de tip baionetă, cu cheie de poziționare (împiedică schimbarea accidentală a polarității) care asigură o bună presiune de contact.

Termocuple, referința se găseşte la 0C

J (fier-constantan)	–210C…+760C, aprox. 50µV/C
K (chromel-alumel)	–270C…+1370C, aprox. 40µV/C
E (chromel-constantan)	–270C…+1000C, aprox. 60µV/C
T (cupru-constantan)	–270C…+400C, aprox. 40µV/C
S (platină-platină,10%rhodiu)	0C…+1760C, aprox. 5…6µV/C
R (platină-platină,13%rhodiu)	0C…+1760C, 5…6µV/C
B (platină-6%rhodiu,platină-30%rhodiu)	0C…+1820C, 2…3µV/10C

Cea mai spinoasă problemă referitoare la interfațarea termocuplelor cu microcontrolerul se reduce la măsurarea corectă a unei tensiuni utilizând modulul AD intern şi implicit la asigurarea nivelului de tensiune corespunzător pe intrare. De exemplu, un termocuplu S sau R funcționând la 1700C, implică măsurarea unei tensiuni de 1700 * 5µV = 8.5mV. Pentru obținerea rezoluției maxime de 1700C/1024 = 1.6C, semnalul trebuie amplificat de 8.5mV * 588 = 4998 mV (aproximativ 5V, CS al AD). Este evident că este nevoie de un amplificator operațional performant având o tensiune de offset de cel puțin 10 ori mai mică decât valoarea livrată de termocuplu şi foarte stabilă cu variația temperaturii ambiante. Dacă dispunem de acest amplificator e minunat, dar dacă acesta lipseşte, sinapsele neuronilor noştrii vor fi încercați cu proiectarea unui amplificator compus din mai multe amplificatoare, având amplificarea globală egală cu 588 dar performanțele individuale ceva mai slabe. Acest lucru va fi posibil datorită scăderii amplificării individuale a amplificatoarelor din lanț, cu efecte benefice asupra driftului termic global.

4.11 Interfațarea circuitului integrat LM135 sau AD22100A

Atât LM135A (National Semiconductors) sau βM135 (IPRS Băneasa) cât şi AD22100A (Analog Devices) sunt senzori de temperatură ambiantă cu ieşire analogică. Singura diferență o reprezintă panta de variație a semnalului de ieşire cu temperatura. Acesta este motivul pentru care, caracteristicile comparative ale celor doi senzori se găsesc în tabelul următor:

Parametru	**LM135A**	**AD22100A**
domeniu temperatură	-55C…+150C	-50C…+150C
eroare maximă la 25C	±1C	±2C
eroare maximă FS	±2.7C	±3.7C
coeficient de temperatură	+10mV/K	(V+/5) *22.5mV
alimentare	400µA…5mA	5V
timp de răspuns în aer	3min	100S
capsula	TO92, SO-8, TO46	TO92, SOIC

Din punct de vedere al structurii interne cei doi senzori se deosebesc radical, LM135 este echivalentul electric al unei diode zenner, având un pin de compensare al caracteristicii de răspuns, în timp ce AD22100 este un senzor de temperatură rezistiv montat într-o semipunte de măsură, tensiunea diferențială fiind preluată de un amplificator diferențial.

Din punctul de vedere al modului de interfațare la microcontroler, cei doi senzori se comportă identic, tensiunea fiind preluată obligatoriu de o intrare de convertor AD. βM135 are tensiunea $U_{0°C}$=2.7315V pentru un curent I_{senzor} = 0.4…5mA, eroarea maximă de măsură la 0C respectiv 100C, cu calibrarea efectuată la 20C, este de ±1.5°C. Calibrarea se face utilizând un potențiometru semireglabil de 10K, conectat în mod potențiometric pe terminalele anod şi catod ale senzorului al cărui cursor este conectat cu pinul de ajustare. Dacă curentul generat în senzor se păstrează constant, interschimbabilitatea senzorilor este dependentă de prezența acestui potențiometru. Variația tensiunii de ieşire a lui AD22100 este cuprinsă între 1375mV pentru 0C şi 3625mV pentru 100C. Ambii senzori necesită amplificare suplimentară dacă se doreşte obținerea rezoluției maxime de măsură, însă pot fi interfațați direct la convertorul AD intern al PIC-ului, pentru măsurători nepretențioase.

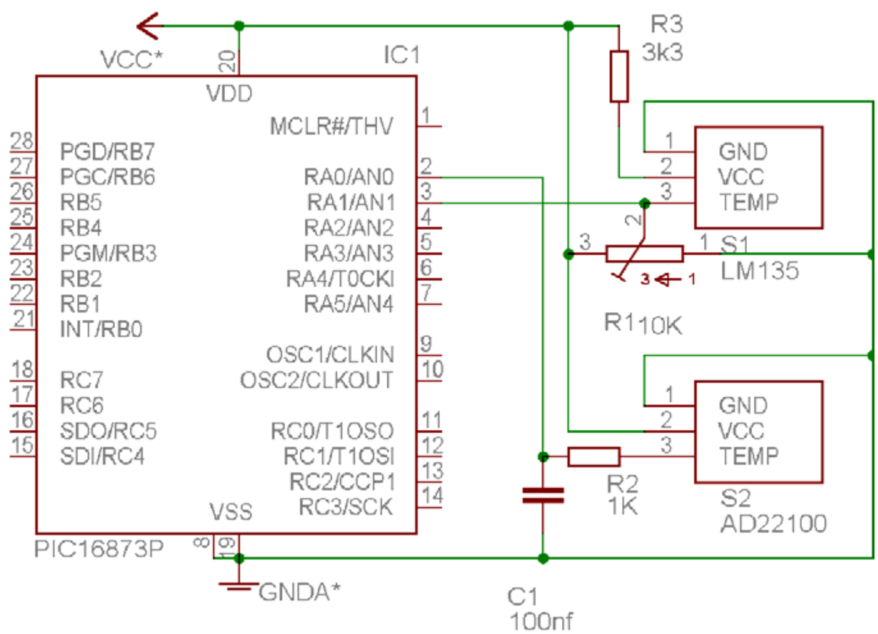


fig.4- 27 Interfațarea directă a senzorilor LM135 şi AD22100

Un astfel de exemplu în care sunt utilizați doar 7 biți din cei 10 posibili oferiți de convertorul AD pentru citirea unui senzor βM135, este sugerat în rutina de mai jos:

```
procedure temperature_read is
-- ----------------------------
 f877_adcon0 = 0b_1000_0001    -- fosc/32, a0, ch0 on
  delay_10uS ( 2 )             -- aşteaptă timpul min. de achiziție
  adcon0_go = high              -- porneşte conversia
  while adcon0_go loop end loop
                                -- aşteaptă terminarea conversiei
```

```
   ch_hi = f877_adresh
   bank_1
   asm movf    f877_adresl,w
   bank_0
   asm movwf   ch_lo
   temperature = ( ch_lo - 0x2e ) / 2    -- 0...127 range; 7 bit res
end procedure

-- ----------------------
-- temperature control
-- ----------------------
  no_int                               -- dezactivează întreruperile
  temperature_read                     -- măsoară temperatura
  if temperature > 100 then avarie q_apa = on
       display_ava                      -- display avarie !
       ava_ret                       -- ieşire din semnalizare avarie
  end if
```

Alegând convenabil valoarea rezistenței R3 (sau poziția cursorului potențiometrului semireglabil R1) pentru temperatura de referință în jurul căreia este necesară măsurarea cu precizie maximă, se poate calcula simplu valoarea care trebuie afişată (şi care reprezintă temperatura reală măsurată), prin scăderea unei constante din tensiunea generată de senzor, urmată de o împărțire întreagă. Eroarea obținută este de cca. 1C în jurul lui 25C şi mai mult de 10C la temperaturi în jur de 100C, însă suficient de bună pentru aplicația considerată drept exemplu.

Bineînțeles că rezoluția poate fi crescută la maxim utilizând un amplificator operațional diferențial şi o referință stabilă de tensiune. Ajustând convenabil polaritatea şi valoarea tensiunii de referință, se poate obține pentru capul de scală al temperaturii de măsurat, valoarea maximă admisă de convertorul AD al PIC-ului. Rezultatul este obținerea unei rezoluții de 10 biți pentru tot domeniul de temperatură, care produce un semnal electric la ieşirea amplificatorului cuprins între 0 şi 5V.

4.12 DS1820, DS1620 termometru digital inteligent interfațat pe bus de 1 fir sau de 3 fire

Familia de senzori inteligenți de temperatură produşi de Dalas Semiconductors este extrem de bogată. Eu am avut plăcerea să lucrez cu două tipuri de senzori: DS18S20 cu ieşire pe bus cu un singur fir şi DS1620 cu ieşire standard pe bus de trei fire. Diferențele interne dintre cei doi senzori fiind minore, diferind doar capsula, algoritmul de împachetare al datelor şi modul de ieşire al acestora spre utilizator, le voi prezenta comparativ în ceea ce urmează.

parametru	**DS1620**	**DS18S20**
temperatura	-55C…+125C	-55C…+125C
rezoluție/precizie	9biți/0.5C sau 0.1C	9biți/ 0.5C sau 0.1C
trigger alarmă	da	da
comunicație	3 fire: data, clk, reset	1 fir: data bidirecționala
autoîncălzire	da	nu
cod serial unic	nu	da 64 biți

parametru	**DS1620**	**DS18S20**
alimentare	2.7V…5.5V	3V…5.5V
capsula	DIP8, SOIC	TO92, SOIC

Ambii senzori au un trigger de alarmare cu două nivele (HI şi LO) pentru valori prestabilite de utilizator. Diferenţa majoră constă în faptul că DS1620 are disponibili la capsulă atât T_{HIGH}, T_{LOW} cât şi T_{COM} ceea ce îi dă posibilitatea de a fi utilizat ca termostat independent (poate controla direct un releu prin intermediul unui buffer) în timp ce DS18S20 trebuie să fie conectat în permanenţă cu microcontrolerul. Precizia de măsură cu calibrare iniţială este de 0.5C însă poate fi crescută la 0.1C cu nişte artificii destul de complexe.

Atât DS1620 cât şi DS18S20 măsoară temperatura numărând impulsurile de tact generate de un oscilator cu un coeficient de temperatură coborât, care trec printr-o poartă logică comandată de un oscilator având un coeficient de stabilitate termică ridicat. Numărătorul corespunzător oscilatorului a cărui frecvenţă este variabilă cu temperatura, este presetat cu o valoare ce corespunde unei temperaturi de –55C. Dacă numărătorul atinge valoarea 0 înainte de terminarea perioadei în care poarta lasă să treacă impulsurile, registrul de temperatură (presetat şi el la –55C) este incrementat, indicând creşterea temperaturii. Acelaşi numărător este încărcat cu valoarea determinată de registrul acumulator al amplificării, care compensează neliniaritatea de tip parabolic a dependenţei frecvenţei oscilatorului cu temperatura prin schimbarea numărului de impulsuri necesar pentru ca numărătorul să fie incrementat cu un grad (conţinutul numărătorului fiind în unităţi de temperatură). Pentru a obţine rezoluţia dorită este nevoie ca ambele valori ale numărătorului şi ale acumulatorului să fie cunoscute pentru o temperatură dată. Astfel registrul acumulator conţine o informaţie de înaltă rezoluţie a temperaturii măsurate, care poate fi citită şi utilizată la compensarea prin metode numerice a rezoluţiei de ieşire, de la 0.5C la 0.1C. Pentru rezoluţia de 0.5C acest calcul este făcut automat în interiorul senzorului. Temperatura rezultată este extrasă în format complement faţă de 2, pe 9 biţi pentru DS1620, respectiv pe doi octeţi pentru DS18S20:

	DS1620		**DS18S20**	
temperatura	ieşire binară	ieşire hexa	ieşire binară	ieşire hexa
+125C	0_1111_1010	00Fah	0000_0000_1111_1010	00Fah
+25C	0_0011_0010	0032h	0000_0000_0011_0010	0032h
+0.5C	0_0000_0001	0001h	0000_0000_0000_0001	0001h
+0C	0_0000_0000	0000h	0000_0000_0000_0000	0000h
-0.5C	1_1111_1111	01FFh	1111_1111_1111_1111	FFFFh
-25C	1_1100_1110	01CEh	1111_1111_1100_1110	FFCEh
-55C	1_1001_0010	0192h	1111_1111_1001_0010	FF92h

Pentru DS1620 data poate fi citită sau scrisă fie ca un cuvânt de 9 biţi prin trecerea pinului RST în stare logică *low* după al nouălea bit, fie ca două cuvinte de 8 biţi pentru care primii 7 biţi ai octetului de semn sunt ignoraţi. Acestă a doua metodă este utilizată şi pentru citirea datelor din DS18S20. Ecuaţia ce permite obţinerea rezoluţiei înalte de măsură, aplicabilă ambilor senzori este:

$$temperature = temp_read - 0.25 + \frac{count_per_c - count_remain}{count_per_c}$$

Unde: *temp_read* este valoarea de temperatură ce apare în tabelul de mai sus dar care este truncată la 0.5C (se pierde un LSB), *count_remain* este valoarea numărătorului după blocarea porţii logice şi poate fi citită cu comanda READ COUNTER, *count_per_c* este valoarea existentă în acumulator şi poate fi citită cu comanda READ SLOPE. Formatul acestei ecuaţii nu este cel mai fericit pentru algoritmul matematic ce se poate implementa cu PIC, motiv pentru care trebuie puţin rearanjată într-o formă care poate fi mult mai folositoare:

temperature = temp_read + 1/2LSB - count_remain / count_per_c

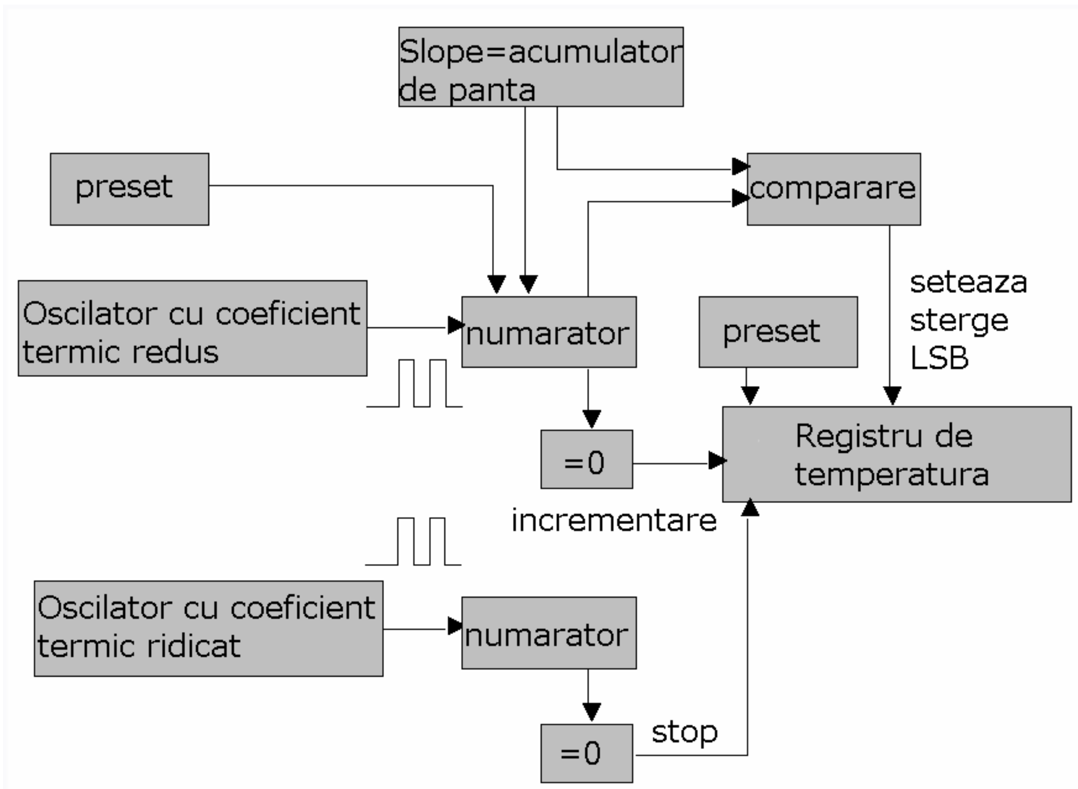


fig.4- 28 Principul de măsură a temperaturii utilizat în DS1620 şi DS18S20

DS620 dispune de registru de configurare numit STATUS REGISTER al cărui biţi sunt:

DONE	THF	TLF	NVB	1	0	CPU	1SHOT

DONE: 1, conversie terminată
0, conversie în progres
THF: 1, t ≥ TH, rămâne neschimbat după setare până la reset
TLF: 1, t ≤ TL, rămâne neschimbat după setare până la reset
NVB: 1, scriere în memoria eeprom în progres (o scriere durează cel puţin 10mS)
0, not busy
CPU: 0, funcţionare independentă, CLK are rol de start conversie când RSTeste low
1, DS1620 comunică cu microcontrolerul
1SHOT: 0, conversie continuă
1, o singură conversie

Setul de instrucţiuni al DS1620 conţine 11 comenzi:

Read temperature [Aah] citeşte ultima valoare de temperatură (9biţi)
Write TH [01h] scrie valoarea temperaturii în registrul TH
Write TL [02h] scrie valoarea temperaturii în registrul TL
Read TH [A1h] citeşte valoarea registrului TH
Read TL [A2h] citeşte valoarea registrului TL
Read counter [A0h] citeşte valoarea numărătorului
Read slope [A9h] citeşte valoarea acumulatorului
Start convert T [Eeh] începe o conversie de temperatură
Stop convert T [22h] termină o conversie; opreşte modul de conversie continuă
Write config [0Ch] scrie în registrul de configurare (status)
Read config [Ach] citeşte valoarea registrului status

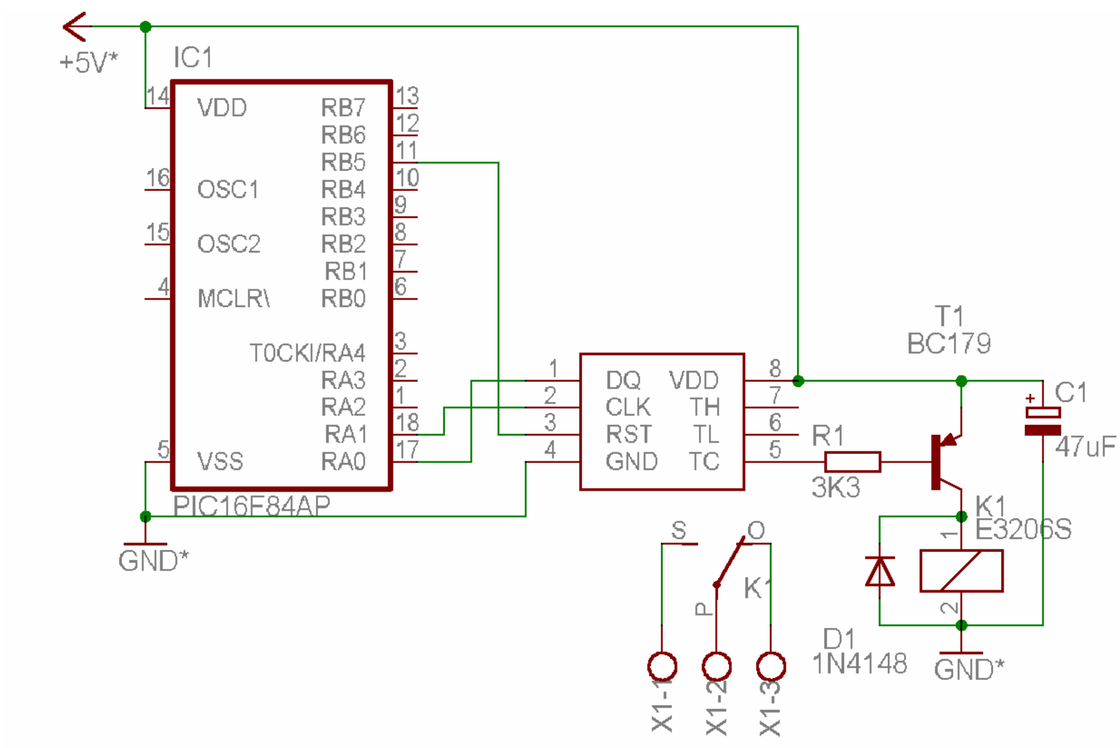


fig.4- 29 Interfaţarea DS1620 la microcontroler cu acţionare de termostat independentă

DS1620 nu pune nici o problemă de interfaţare cu microcontrolerul, certitudine rezultată din programul de mai jos, realizat într-un lung weekend ploios:

```
; biblioteca pentru comunicaţie pe 3 fire, testată pe DS1620
include 16f628_4
include jpic628
include jdelay
```

```
include ds1620p
include hd447804
include jprint

pragma target fuses 0x_3fc1       ; configurează fuse-urile
cmcon = 0x07                      ; dezactivează comparatorul

var volatile bit data ; pin_a0 is data
var volatile bit clk is pin_a1
pin_a1_direction = output
var volatile bit reset is pin_b5
pin_b5_direction = output

procedure out_3w ( byte in data_3w ) is ; trimite data serial
 pin_a0_direction = output
  for 8 loop
   assembler
    bcf   clk              ; clk low
    bcf   data             ; data low
    btfsc data_3w, 0       ; dată validă pentru transmis ?
    bsf   data             ; dacă da trimite data
    rrf   data_3w, f        ; serializează, lsb este primul
    bsf   clk              ; clk high
   end assembler
  end loop
end procedure

procedure in_3w  ( byte out data_3w ) is ; recepţionează serial data
pin_a0_direction = input
 for 8 loop
   assembler
    bcf   clk           ; clk low
    bcf   status_c      ; curăţă status pentru următoarea verificare
    btfsc data          ; verifică dacă vine o dată validă
    bsf   status_c      ; dacă da, prepară ultimul bit
    rrf   data_3w, f    ; şi roteşte-l în data_3w
    bsf   clk           ; clk high
   end assembler
 end loop
end procedure

; citeşte cuvântul de configurare
procedure read_config ( byte out config_status ) is
 reset = high
 out_3w ( 0x_ac )
 in_3w  ( config_status )
 reset = low
end procedure

; scrie cuvântul de configurare
procedure write_config ( byte in new_config_status ) is
 reset = high
 out_3w ( 0x_0c )
 out_3w ( new_config_status )
```

```
 reset = low
end procedure

; porneşte conversia
procedure start_convert is
 reset = high
 out_3w ( 0x_ee )
 reset = low
end procedure

; opreşte conversia
procedure stop_convert is
 reset = high
 out_3w ( 0x_22 )
 reset = low
end procedure

; încarcă numărătorul ( pentru mod de înaltă rezoluţie )
procedure load_counter is
 reset = high
 out_3w ( 0x_41 )
 reset = low
end procedure

; cireşte numărătorul ( pentru mod de înaltă rezoluţie )
procedure read_counter ( byte out count_lsb, byte out count_msb ) is
 asm bsf reset
 out_3w ( 0x_a0 )
 in_3w ( count_lsb )
 in_3w ( count_msb )
 asm bcf reset
end procedure

procedure read_temperature ( byte out temp_lsb, bit out sign,
                             bit out half_degree ) is
 asm bsf reset            ; reset high
 out_3w ( 0x_aa )         ; comandă specifică
 in_3w  ( temp_lsb )      ; citeşte lsb
 asm bcf status_c         ; curăţă carry
 asm rrf temp_lsb, f      ; temp = temp/2, jumatea de grad în carry
  if status_c then asm bsf half_degree
                          ; setează bitul half_degree
  else asm bcf half_degree end if
 var byte temp_msb
 in_3w  ( temp_msb )          ; citeşte msb
 asm bcf reset                ; reset low
 asm bcf status_c             ; clear c
 asm rrf temp_msb, f          ; adu semnul în carry şi
 if status_c then sign = high ; testează, dacă are valoare negativă converteşte
  asm comf temp_msb, f        ; complementează,
  asm incf temp_msb, f        ; şi incrementează = valoare pozitivă
  else sign = low end if      ; valoare pozitivă, nemodificată
end procedure
```

```
; scrie valoarea termostatului
procedure write_tx ( byte in l_h, byte in tx_low,
                                 byte in tx_sign ) is
; l_h = 0x_01 for TH ( high temperature register )
; l_h = 0x_02 for TL ( low temperature register )
 reset = high
 out_3w ( l_h )            ; adresa low sau high a termostatului
 out_3w ( tx_low )         ; scrie lsb
 out_3w ( tx_sign )        ; scrie msb  -> doar semnul
 reset = low
end procedure

; citeşte valoarea termostatului doar pentru test
procedure read_tx ( byte in l_h, byte out tx_low, byte out tx_sign )
is
; l_h = 0x_a1 pentru TH
; l_h = 0x_a2 pentru TL
; tx_sign = 0  temperatură pozitivă
; tx_sign = 1  temperatură negativă
; tx_low în concordanță cu fila de catalog a dispozitivului
 asm bsf reset
 out_3w ( l_h )            ; adresa low sau high a termostatului
 in_3w ( tx_low )           ; citeşte lsb
 in_3w ( tx_sign )          ; citeşte semnul
 asm bcf reset
end procedure

var bit sign, half_degree
var byte  temp_lsb, TH, TL, degree_lo, degree_hi, count_lo, count_hi
var volatile byte config_status
; citeşte numărătorul pentru modul hi_res
procedure hi_res_read is
 read_counter ( count_lo, count_hi )
 load_counter
 read_counter ( degree_lo, degree_hi )
end procedure

hd44780_clear
write_config ( 0x_0A )      ; comunicație cu cpu, conversie continuă
delay_1mS ( 10 )            ; necesar pentru scrierea în eeprom
write_tx ( 0x_01, 0x_2e, 0x_00 ) ; TH = +23C, termostat high
delay_1mS ( 10 )
write_tx ( 0x_02, 0x_2a, 0x_00 ) ; TL = +21C, termostat low
delay_1mS ( 10 )
start_convert               ; porneşte conversia
 forever loop
; read_config ( config_status ) ;testăm funcționarea corectă
 read_temperature ( temp_lsb,sign,half_degree )
                            ; citeşte temperatura şi semnul
 hd44780_line1
 if sign then hd44780 = "-" else hd44780 = "+" end if
 hd44780_position1 ( 1 )
 print_decimal_2 ( hd44780, temp_lsb, "0" )
 hd44780 = "."
```

```
 if half_degree then
  hd44780 = "5"
 else hd44780_position1 ( 4 ) hd44780 = "0"
 end if
 hd44780 = 223              ; simbolul °, pătrăţos dar gata definit…
 hd44780 = "C"
 delay_100mS ( 3 )           ; întârziere pentru afişare
end loop
```

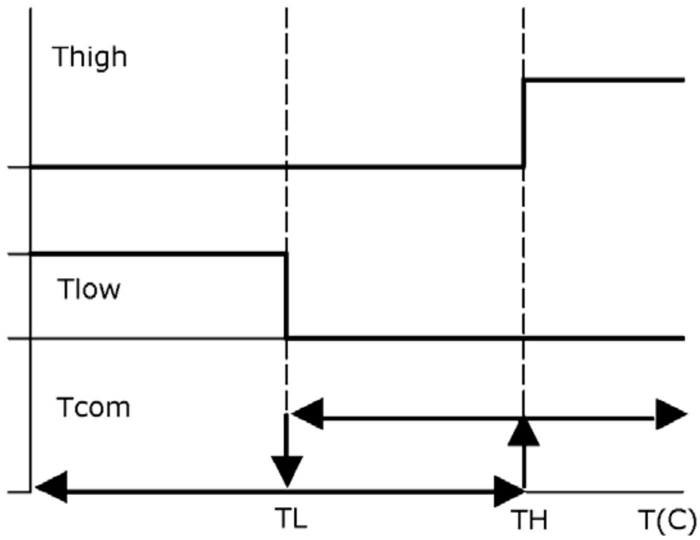


fig.4- 30 Diagrama de histerezis a ieşirii de termostat

Prorămelul de mai sus realizează afişarea temperaturii ambiante utilizând modul de afişare cu rezoluţie de 0.5C, pe un afişaj LCD din cele prezentate în subcapitolul 4.1.3. Funcţia de termostat a DS1820 este de asemenea implementată, termostatul va funcţiona între limitele TH = 23C respectiv TL = 21C conform algoritmului cu histerezis din figura 4.30

Arhitectura internă a lui DS18S20 este ascunsă în spatele unui *scratchpad* (fig.4-31) (matrice de memorie reinscriptibilă) fiind identică cu cea descrisă în fig.4.28

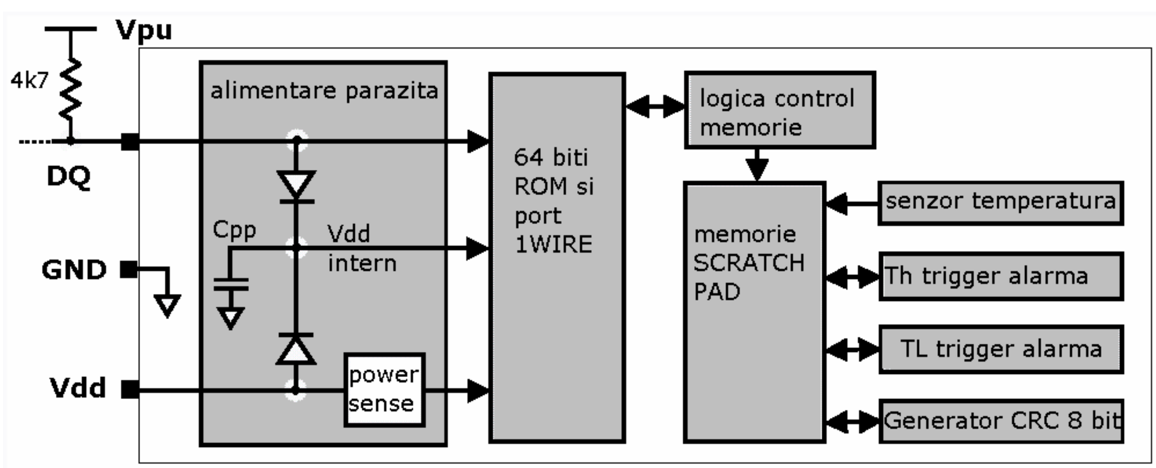


fig.4- 31 Schema bloc a senzorului de temperatură DS18S20

Blocul de alimentare cu tensiune “parazită”, fură energie de alimentare din semnalul ce se vehiculează pe intrarea/ieşirea de date DQ când aceasta se găseşte în stare logică *high*. Condensatorul intern Cpp se încarcă în acest moment şi menține tensiunea de alimentare când DQ este *low*. Desigur că opțiunea de alimentare de la o sursă exterioară rămâne valabilă, însă sunt situații când alimentarea cu doar două fire este benefică, în detrimentul utilizării unui software mai complicat. Deoarece înscrierea datelor din *scratchpad* în eeprom consumă cca. 1.5mA, este necesară utilizarea unei rezistențe de *pull_up* suficient de mici pentru a asigura acest curent când au loc operații interne de conversie sau scriere în eeprom. Pentru a putea conecta mai multe dispozitive pe acelaşi bus este nevoie de un mijloc de identificare a senzorului a cărui temperatură se citeşte. Codul ROM de 64 de biți realizează acest lucru. El se compune din:

(MSB) 8 bit CRC	**48 bit număr de serie**	**8bit codul familiei[10h] (LSB)**

Microcontrolerul trebuie să recalculeze CRC-ul şi să-l compare cu valoarea memorată în senzor utilizând generatorul polinomial din figura următoare, algoritmul fiind implementat în funcția *d1w_read_byte_with_CRC*:

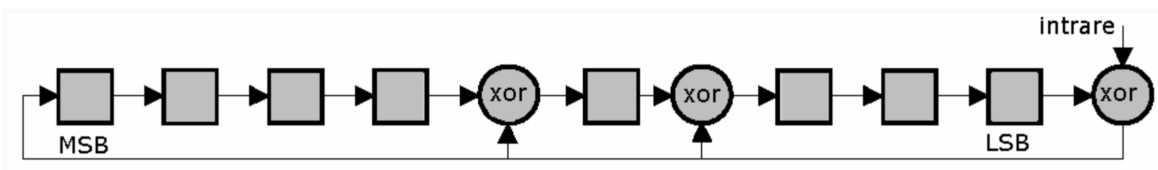


fig.4- 32 Generatorul polinomial de refacere al CRC

Memoria scratchpad are 8 octeți (fig. 4.34). Setul de instrucțiuni ce jonglează cu datele memorate aici diferă de cele utilizate de DS1620.

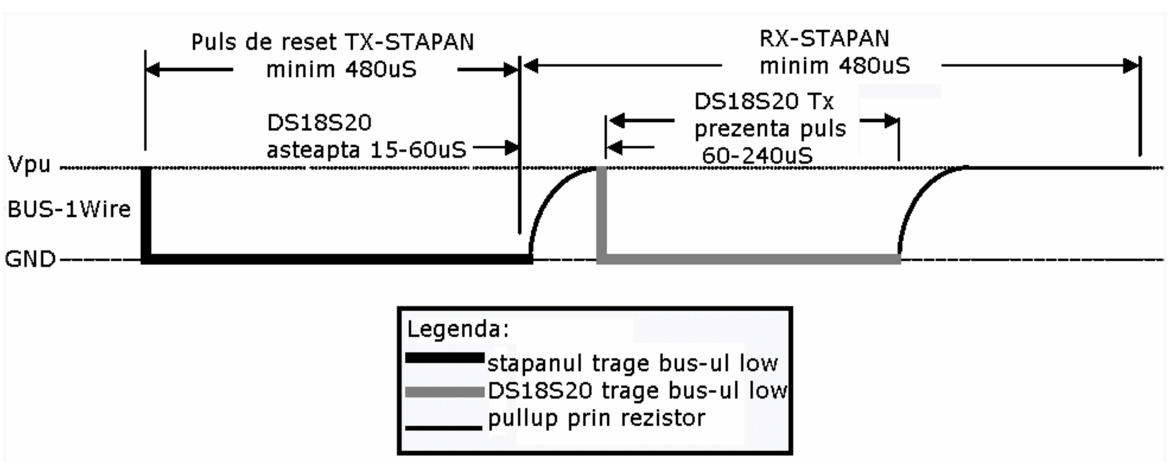


fig.4- 33 Inițializarea DS18S20

Inițializare: secvență de pulsuri transmisă pe bus de master urmată de prezența pulsurilor transmise de slave (fig4.33)
Comenzi ROM, operează cu codul unic de 64 biți:

Search ROM [F0h] identifică numărul de sclavi conectaţi la bus şi tipul acestora
Read ROM [33h] citeşte codul de 64 biţi pentru un singur senzor existent pe bus
Match ROM [55h] comanda, urmată de 64 biţi (adresa sclavului)permite masterului să adreseze un sclav specific
Skip ROM [CCh] în urma acestei comenzi, masterul adresează toţi senzorii conectaţi pe bus. Pentru un singur senzor prezent pe bus, următoarea comandă trebuie să fie **Read Scratchpad.**
Alarm Search [ECh] comandă similară cu **Search ROM**, va răspunde doar sclavul cu bitul de alarmă setat

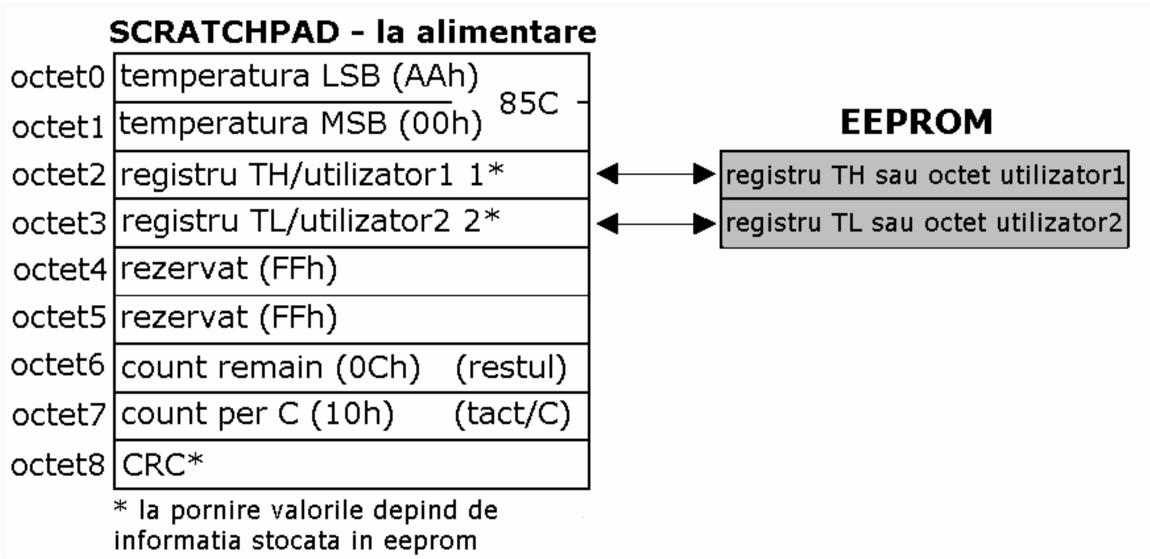


fig.4- 34 Conţinutul memoriei "scratchpad"; LSB conţine valoarea temperaturii iar MSB semnul acesteia

Comenzi funcţionale (comenzi de citire-scriere şi iniţiere a conversiei):

Convert T [44h] iniţiază o singură conversie de temperatură. Rezultatul este scris în cei doi regiştrii: temperature LSB şi tempearature MSB. Dacă senzorul se găseşte în modul cu alimentare parazită, PIC-ul trebuie să asigure o rezistenţă de pull-up suficient de scăzută, cel puţin 10uS după primirea comenzii. In modul de alimentare cu sursă externă, DS18S20 va răspunde trimiţând 0 pentru o conversie de temperatură aflată în progres, respectiv 1 pentru o conversie terminată.
Write scratchpad [4Eh] doi biţi sunt scrişi în *scratchpad*, primul în TH şi al doilea în TL, LSB este trimis primul.
Read scratchpad [BEh] Masterul citeşte conţinutul *scratchpad*-ului pornind cu octetul 0 LSB şi terminând cu octetul 9.
Copy scratchpad [48h] regiştrii TH şi TL sunt copiaţi în eeprom, necesită cel puţin 10uS de pull-up suficient de redus.
Recall [E2h] citeşte TH şi TL din eeprom şi îi rescrie în *scratchpad*, DS18S20 transmite 0 cât timp acţiunea este în desfăşurare şi 1 după terminarea acesteia.

Read power supply [B4h] comandă utilizată pentru detectarea senzorilor de pe bus ce folosesc modul de alimentare parazită. Pe durata *read time slots* bus-ul va fi, fie în stare *low* (stare generată de senzorii alimentați parazit) fie *high* (stare generată de senzorii cu alimentare independentă) conform fig.4-35.

In ceea ce priveşte algoritmul de măsură al temperaturii cu DS18S20 se întrevăd câteva mici probleme: deoarece există posibilitatea conectării mai multor senzori pe acelaşi bus, utilizatorul trebuie întâi să cunoască codul unic de 64 de biți al fiecărui senzor utilizat. Programul care citeşte acest cod şi îl transmite PC-ului pe interfața serială, utilizând comunicația supervizată de USART-ul intern (PIC16F628x sau PIC16F87x) este următorul:

```
include 16f628_20
include jpic628
include usart    ; bibliotecă de comunicatie serială, cap.6
include jprint   ; bibliotecă standard de conversie

var byte data
var bit GOOD_crc      ; bitul de memorare al rezultatului calculului CRC
var volatile bit d1w_bus_in    is pin_b5
var volatile bit d1w_bus_out   is pin_b5_direction
d1w_bus_out  = high ; pinul PIC devine intrare menținută high prin pull-up
```

➢ După definirea variabilelor globale, programul se compune dintr-o serie de rutine structurate la nivel de bit şi octet pentru scriere şi citire pe bus:

```
procedure d1w_write_bit( bit in x ) is    ; scrierea unui bit pe bus prin:
   d1w_bus_out = low                       ; pinul este ieşire
   delay_10us( 1 )                         ; timp de 10uS
   if x == high then d1w_bus_out = high    ; testează bitul citit de pe bus  şi menține...
   end if                                  ; direcția intrare
   delay_10us( 8 )                         ; menține bus-ul high 80uS
   d1w_bus_out = high                      ; dacă bitul a fost low, trece bus-ul high
   delay_10us( 1 )                         ; şi menține 10uS
end procedure

procedure d1w_write_byte( byte in b ) is  ; scrierea unui octet pe bus
   var bit x at b : 0                      ; lsb primul
   for 8 loop                              ; de 8 ori
       d1w_write_bit( x )                  ; scrie bit cu bit,
      b = b >> 1                           ; roteşte dreapta…
   end loop                                ; întregul octet
end procedure

procedure d1w_read_bit( bit out x ) is    ; citirea unui bit
   x = high
   d1w_bus_out = low                       ; inițializare 10uS
   delay_10us( 1 )
   d1w_bus_out = high                      ; pinul devine intrare
   asm nop asm nop                         ; scurtă întârziere de 0.4uS (20MHz)
   if d1w_bus_in == low then x = low
```

```
   end if                              ; adu valoarea lui x în concordanță cu bus-ul
   delay_10us( 7 )
end procedure

procedure d1w_read_byte( byte out c ) is ; citirea unui octet de pe bus
   var bit x at c : 7                    ; definirea bitului msb al octetului
   for 8 loop                            ; generarea cuvântului de 8 biți, bit cu bit
      c = c >> 1                         ; rotire dreapta
      d1w_read_bit( x )                  ; şi citire
   end loop
end procedure
```

➢ Apoi sunt definite instrucțiunile specifice senzorului DS18S20:

```
procedure d1w_reset is     ; secvență de reset
   d1w_bus_out = low
   delay_10us( 70 )
   d1w_bus_out = high
   delay_10us( 70 )
end procedure

procedure DS1820_start_temperature_conversion is
   d1w_reset
   d1w_write_byte( 0xCC )
   d1w_write_byte( 0x44 )
end procedure

procedure DS1820_read_temperature_raw( byte out h, byte out l ) is
   d1w_reset
   d1w_write_byte( 0xCC )
   d1w_write_byte( 0xBE )
   d1w_read_byte( l )
   d1w_read_byte( h )
end procedure
```

➢ Pentru afişarea simplificată a datelor sub controlul unui program terminal pe PC, este folosită o procedură ce ascunde rutina de transmisie a usart.jal într-o pseudovariabilă utilizată de o procedură *put.* Aceasta deoarece funcționarea modulului USART va fi explicată doar în capitolul 6 în timp ce procedura de tip *put* sau *get* a fost deja analizată în capitolul 2.

```
--  pseudo-variabilă utilizată de usart cu jprint.jal
procedure async_tx_usart'put( byte in value ) is
   async_tx( value )
end procedure
```

➢ Funcția următoare realizează citirea a 8 sau 9 octeți reprezentând seria senzorului respectiv temperatura şi calculează crc-ul, dacă transmisia a fost corectă, valoarea octetului calculat va fi 0, respectiv bitul returnat va fi în stare *low*

```
var byte d1, d2, d3, d4, d5, d6, d7, d8, d9
var byte GOOD_crc
```

```
function d1w_read_byte_with_CRC( byte in nr_byte )return bit is

   var byte bb = 0, n = 0 , crcbyte = 0    -- reset inițial
-- biții necesari reconstrucției prin generator polinomial ( fig4.32 )
   var bit  bb_bit0 at bb : 0
   var bit  crcbyte_bit0 at crcbyte : 0 ,
            crcbyte_bit2 at crcbyte : 2
   var bit  crcbyte_bit3 at crcbyte : 3 ,
            crcbyte_bit7 at crcbyte : 7
   var bit  crcbit

for nr_byte loop   -- 8 octeți pentru readrom ID,
                   -- 9 octeți pentru citirea temperaturii
     d1w_read_byte( bb )
   n = n + 1              -- n = 1
   if n == 1 then d1 = bb end if
                          -- d1…d9 sunt aici variabile globale
   if n == 2 then d2 = bb end if
   if n == 3 then d3 = bb end if
   if n == 4 then d4 = bb end if
   if n == 5 then d5 = bb end if
   if n == 6 then d6 = bb end if
   if n == 7 then d7 = bb end if
   if n == 8 then d8 = bb end if
   if n == 9 then d9 = bb end if

-- ---calculul crc------------------------------------------
  for 8 loop
   crcbit = crcbyte_bit0 ^ bb_bit0       ;  bit0 sau exclusiv cu LSB
   crcbyte = crcbyte >> 1                ;  generarea octetului crcbyte
   crcbyte_bit7 = crcbit                 ;  refacerea bitului 7
   crcbyte_bit2 = crcbyte_bit2 ^ crcbit  ;  bit2 sau exclusiv cu crcbit
   crcbyte_bit3 = crcbyte_bit3 ^ crcbit  ;  bit3 sau exclusiv cu crcbit
   bb = bb >> 1                          ;  şiftare necesară !
  end loop
-- ---------------------------------------------------------
end loop

if crcbyte == 0 then crcbit = true  else crcbit = false
end if                            --  crcbyte = 0; nu este eroare CRC
return crcbit
end function

   d1w_reset
   d1w_write_byte( 0x33 )         -- READROM ID
   GOOD_crc = d1w_read_byte_with_CRC( 8 )
-- afişează cei 8 octeți ai codului unic pentru DS18S20

async_tx( "I" ) async_tx( "D" )async_tx( " " )
print_hexadecimal_2( async_tx_usart , d1, "0" ) async_tx( " " )
print_hexadecimal_2( async_tx_usart , d2, "0" ) async_tx( " " )
print_hexadecimal_2( async_tx_usart , d3, "0" ) async_tx( " " )
print_hexadecimal_2( async_tx_usart , d4, "0" ) async_tx( " " )
```

```
print_hexadecimal_2( async_tx_usart , d5, "0" ) async_tx( " " )
print_hexadecimal_2( async_tx_usart , d6, "0" ) async_tx( " " )
print_hexadecimal_2( async_tx_usart , d7, "0" ) async_tx( " " )
print_hexadecimal_2( async_tx_usart , d8, "0" ) async_tx( " " )

if GOOD_crc == true then data = "Y" else data = "N" end if
async_tx( "C" ) async_tx( "R" ) async_tx( "C" )
async_tx( "=" ) async_tx( data )async_tx( " " ) async_tx( 13 )
```

- Dacă a avut loc o citire corectă rezultatul vizibil pe screen-ul programului terminal va fi:

ID 10 AB 1A 3B 00 00 00 42 CRC Y , unde 10 AB 1A 3B 00 00 00 42 reprezintă seria senzorului respectiv, diferită pentru fiecare senzor în parte (nu vă aşteptaţi să-l vedeţi pe acesta!). Observaţi că async_tx_usart este o pseudovariabilă utilizată de print_hexadecimal_2 în timp ce async_tx este procedura standard de transmisie USART. In acest moment, cunoscând valoarea seriei fiecărui senzor, putem scrie rutina de identificare a lor. Pentru doi senzori DS18S20 aceasta poate fi următoarea:

```
var byte sensor_number
var byte name1, name2, name3
; numele senzorilor au trei caractere ASCII

procedure MatchRom( byte in probe_number ) is
   d1w_reset
   d1w_write_byte( 0x55 )

-- adresarea unui singur senzor odată:
  if sensor_number == 1 then
   d1 = 0x_10 -- ID 10 AB 1A 3B 00 00 00 42
   d2 = 0x_AB d3 = 0x_1A d4 = 0x_3B d5 = 0x_00
   d6 = 0x_00 d7 = 0x_00 d8 = 0x_42
   name1 = "B" name2 = "0" name3 = "1"
end if

  if sensor_number == 2 then
   d1 = 0x_10 -- ID 10 A4 F2 3A 00 00 00 0D
   d2 = 0x_A4 d3 = 0x_F2 d4 = 0x_3A d5 = 0x_00
   d6 = 0x_00 d7 = 0x_00 d8 = 0x_0D
   name1 = "B" name2 = "0" name3 = "2"
  end if

   d1w_write_byte( d1 )    -- trimite ID-ul
   d1w_write_byte( d2 )
   d1w_write_byte( d3 )
   d1w_write_byte( d4 )
   d1w_write_byte( d5 )
   d1w_write_byte( d6 )
   d1w_write_byte( d7 )
   d1w_write_byte( d8 )
end procedure
```

- Şi în final putem scrie programul principal, care va demara conversia, va adresa ciclic cei doi senzori pe rând, apoi va citi conţinutul *scratchpad* şi va afişa regiştrii d2 (octetul

cel mai semnificativ adică semnul temperaturii) şi d1 rotit la dreapta cu un bit, sau dacă doriți împărțit cu doi (adică valoarea întreagă a temperaturii)

```
forever loop

DS1820_start_temperature_conversion
sensor_number = sensor_number + 1 ; incrementare adresă senzor
if sensor_number > 2 then sensor_number = 1 end if
   MatchRom( probe_number )   -- adresarea unui singur senzor
   d1w_write_byte( 0xBE )     -- citeşte temperatura ( read scratchpad )
   GOOD_crc = d1w_read_byte_with_CRC( 9 )    -- 9 biți
print_hexadecimal_2( async_tx_usart , d1, "0" ) async_tx( " " ); mso
print_hexadecimal_2( async_tx_usart , d2, "0" ) async_tx( " " ); lso
print_hexadecimal_2( async_tx_usart , d3, "0" ) async_tx( " " ); TH
print_hexadecimal_2( async_tx_usart , d4, "0" ) async_tx( " " ); TL
print_hexadecimal_2( async_tx_usart , d5, "0" ) async_tx( " " ); res
print_hexadecimal_2( async_tx_usart , d6, "0" ) async_tx( " " ); res
print_hexadecimal_2( async_tx_usart , d7, "0" )
                                      async_tx( " " )   ; CR
print_hexadecimal_2( async_tx_usart , d8, "0" )
                                      async_tx( " " )   ; CC
print_hexadecimal_2( async_tx_usart , d9, "0" )
                                      async_tx( " " )   ; CRC
async_tx( " " )
                              ; semnificația d1…d9 din fig4.33
if GOOD_crc == true then data = "Y" else data = "N" end if
async_tx( "C" ) async_tx( "R" )async_tx( "C" ) async_tx( "=" )
async_tx( data ) async_tx( " " )
async_tx( name1 ) async_tx( name2 )async_tx( name3 ) async_tx( ":" )
async_tx( " " )
print_decimal_2( async_tx2 , d1/2 , "0" )async_tx( 13 )
print_decimal_2( async_tx2 , d2 , "0" )async_tx( 13 )
   delay_1s
end loop
```

După cum ați observat, modul de alimentare al celor doi senzori conectați pe bus-ul comun din exemplul precedent era alimentarea separată. Este intuitiv faptul că alimentarea din sursă externă de tensiune creează probleme mult mai mici decât alimentarea parazită, prin "furt" de energie din semnalul de date. Insă după ce programul testat cu senzorul alimentat extern merge fără erori, se poate trece la modul parazit. Este importantă lungimea bus-ului şi valoarea rezistenței de pull-up care poate fi modificată în limite largi (1k5…4k7). Producătorul recomandă creşterea curentului de pull-up prin scăderea rezistenței în timpul conversiei sau al scrierii în eepromul intern al senzorului.. Testarea bus-ului, citirea şi scrierea pe bus, trebuie făcute conform cu marjele de timp impuse de *time slot*. Orice pin al portului B al PIC-ului, cu setarea corespunzătoare din registrul OPTION poate reduce valoarea rezistenței externe prin conectarea în paralel cu aceasta a rezistenței de pull-up interne PIC-ului.

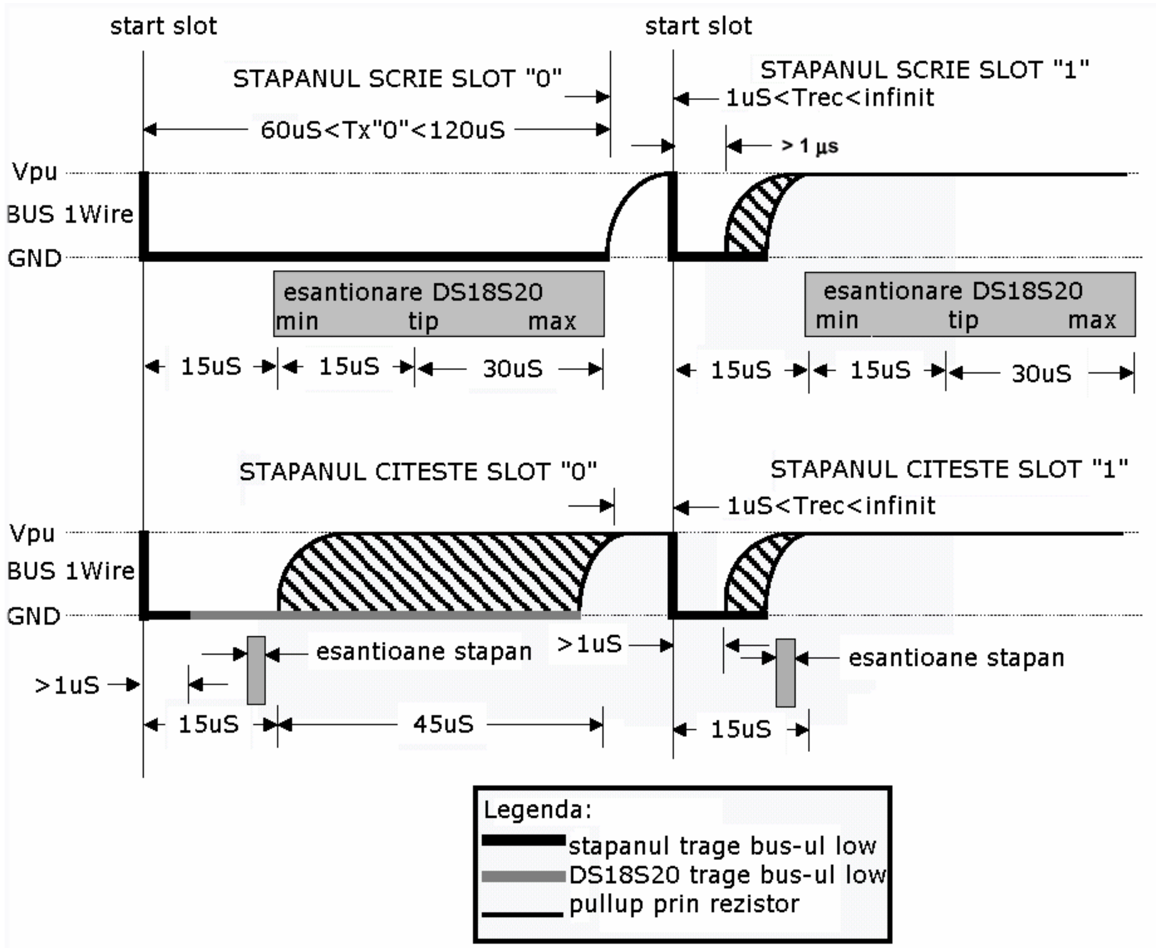


fig.4- 35 Intervalele de timp obligatorii pentru citire şi scriere în DS18S20 *(read-write)*

Graficul din fig.4-35 poate fi completat cu doar câteva cuvinte:

- Timpul este diferit la scrierea unui 1 logic de către PIC sau sau a unui 0 logic de către DS18S20 pe bus; Pic-ul inițiază ambele tipuri de scriere/citire prin trecerea bus-ului în 0 logic pentru cel puțin 1uS:

Write 1 = inițializarea scrierii şi eliberarea bus-ului (bus-ul în 1 logic) timp de 15uS
Write 0 = inițializarea scrierii şi menținerea bus-ului în stare 0 logic timp de 60uS
Read 0 sau *read 1* = inițializarea citirii şi eliberarea bus-ului. DS18S20 va începe transmisia unui 1 (lasă bus-ul liber) sau 0 (trage bus-ul la masă). Datele scrise pe bus sunt valide doar 15uS după frontul căzător al semnalului de inițializare, de aceea PIC-ul trebuie să elibereze bus-ul şi să citescă starea acestuia în primele 15uS de la generarea *time slot*-ului.

- Fereastra de scriere în DS18S20 are loc după 15uS de la inițierea scrierii şi poate dura până la 60 uS
- Durata minimă la scriere/citire este pentru orice situație minimum 60uS, cu o durată de revenire între două scrieri de minim 1uS.

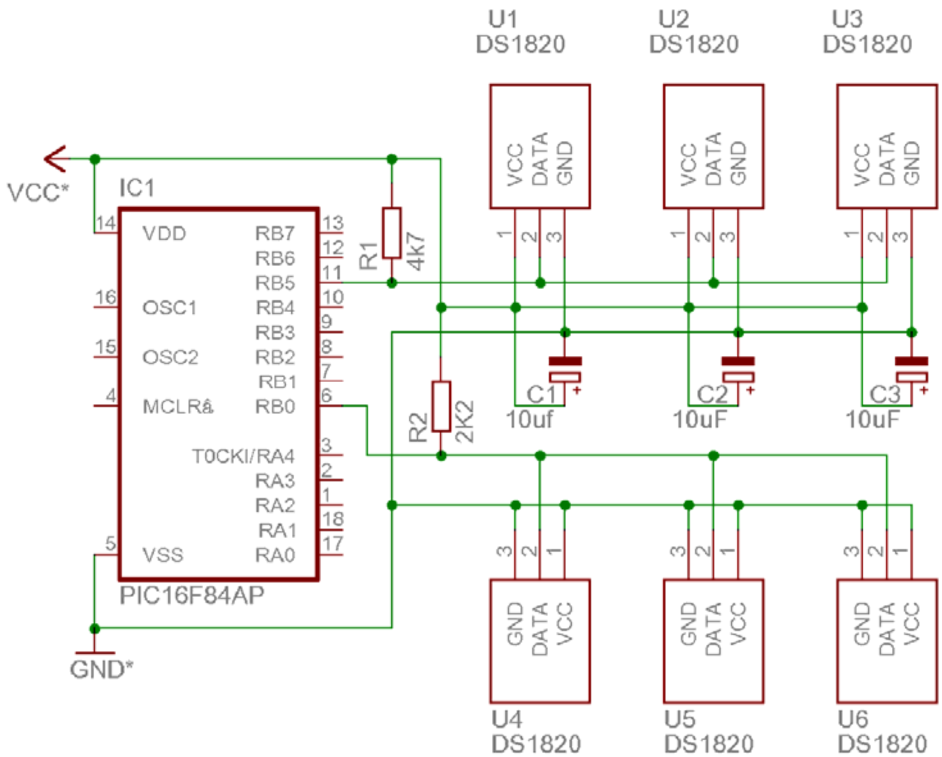


fig.4- 36 Conectarea mai multor senzori DS18S20 pe bus cu alimentare separată şi parazită

Schema de conexiune pe bus de 1 fir sau 3 fire, cu şi fără alimentare parazită este cea din fig.4-36. Dacă bus-ul cu alimentare separată permite filtrarea zgomotului indus în linie datorită lungimii acestuia, bus-ul cu alimentare parazită trebuie realizat cu fir torsadat pentru minimizarea zgomotului şi menţinerea capacităţii parazite a liniei în limite convenabile.

4.13 Un ceas cu termometru la îndemâna oricui !

Una din curiozităţile oricărui mare oraş este ceasul electronic situat în pieţele sau intersecţiile mai importante. Cel mai interesant lucru când mergi la servici este să verifici ce prostie mai arată, fie când afişează ora exactă care este inexactă… fie când afişează o temperatură total anormală pentru anotimpul în care te găseşti. Ei bine, un ceas ce afişează ora exactă şi temperatura şi se află la tine pe birou este la fel de folositor…

Parcurgând capitolele anterioare aveţi la îndemână toate elementele componente necesare acestui proiect, o mână de ajutor fiind acordată în continuare. Schema din imaginea următoare ar trebui să vă fie deja familiară. Noutatea introdusă faţă cele discutate anterior, ar fi modul de alimentare cu baterie de back-up al PIC-ului. Singura precauţie ce trebuie luată este că acumulatorul BAT1 trebuie să fie deja încărcat la introducerea sa în circuit, iar PIC-ul trebuie să aibă BOR inactiv, altfel va rămâne permanent în reset. Se poate utiliza cu succes un acumulator recuperat de pe o placă de bază de PC (*motherboard*). Pentru a obţine precizia mai bună de 5S/lună deviaţie de la timpul etalon este obligatorie

măsurarea reală a frecvenței generate de oscilator. Dacă utilizatorul dispune de un oscilator extern de precizie (este în capsulă metalică cu alimentare independentă şi are înscris un număr de 6 zerouri după virgulă) indiferent de valoarea acestuia, poate aplica cu succes algoritmul descris în cap.3.4.2. Dacă nu este disponibil un astfel de oscilator, se poate măsura frecvența oscilatorului utilizând un repetor (74SN407 sau inversor 74SN404) cu rol de buffer între ieşirea oscilatorului şi intrarea frecvențmetrului, pentru a nu perturba oscilatorul cu capacitatea parazită a sondei de măsură. In fine, dacă utilizatorul nu are nici frecvențmetru, (poate începe prin a şi-l construi singur studiind schemele prezentate în CD:\pic16F84_applications\frequency_meters…) atunci, poate utiliza un algoritm iterativ de modificare a regiştrilor notați roman_x în programul inclus pe CD în directoarea CD:\jal_applications\clock_thermometer, până la obținerea preciziei respective. Aveți răbdare dacă utilizați această metodă ! E nevoie de cel puțin trei-patru reglaje consecutive la interval de patru-cinci zile, cu urmărirea efectului modificării prin compararea cu un ceas etalon.

Nu pot să vă urez altceva decât succes, dacă ați lucrat corect atunci veți obține un dispozitiv cu acuratețe comparabilă cu semnalul radio de ceas provenit din Germania, diferențele lunare maxime între ceasul construit de dvs. şi orice *radio-clock* ce funcționează corect vor fi sub 5 secunde.

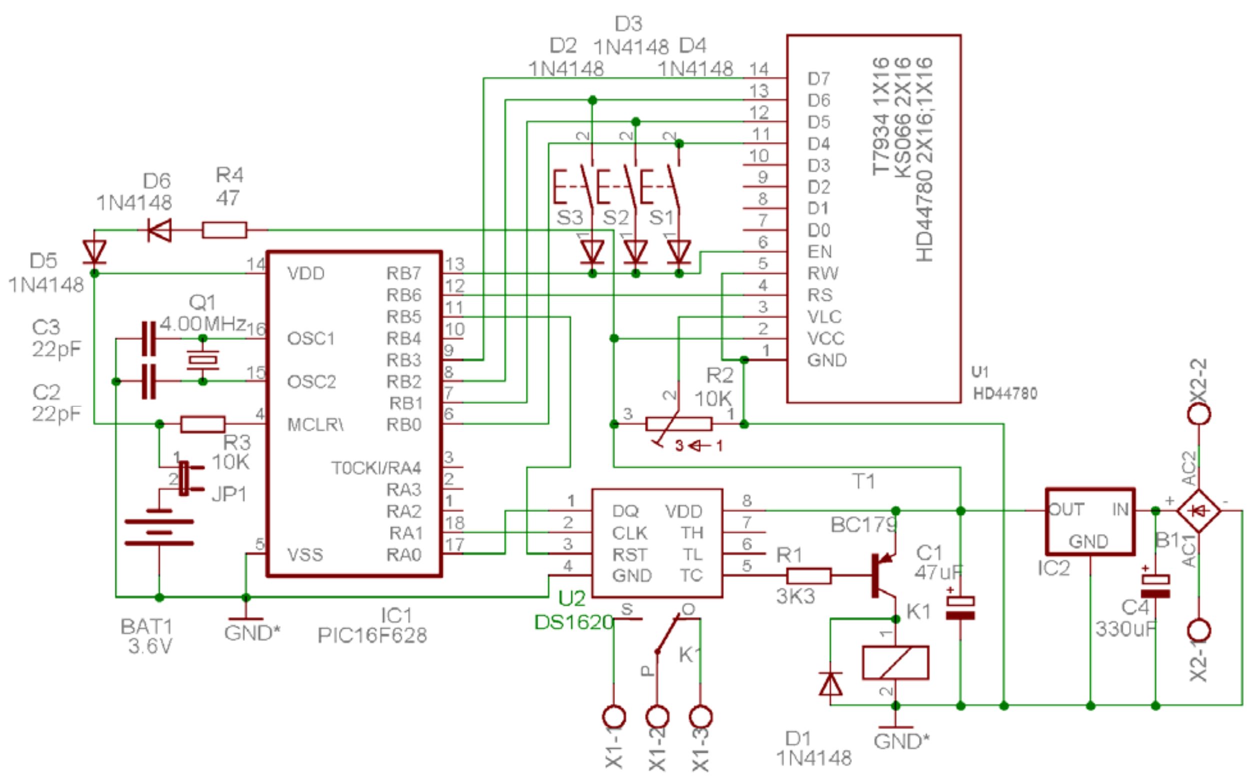


fig.4- 37 Ceas de mare precizie şi termometru termostat cu PIC16F628 şi DS1620

Bibliografie:

1. Utilizarea LCD HD44780, http://www.epemag.wimborne.co.uk/resources.htm
2. Setul de instrucțiuni HD44780, http://www.doc.ic.ac.uk/~ih/doc/lcd/instruct.html
3. DS1620 datasheet, 1999, http://www.dalsemi.com
4. DS18s20 datasheet, http://www.dalsemi.com
5. Grup de lucru via email, http://groups.yahoo.com/group/jallist
6. βM135, filă de catalog, IPRS Băneasa
7. AD22100A, datasheet, Analog Devices
8. MAX121, 308 ksps ADC with DSP interface, datasheet, 1993
9. MAX132, ±18bit ADC with serial interface, datasheet, 1995
10. MCP3202, datasheet, Microchip, http://www.microchip.com
11. SN74HC595, datasheet, Texas Instruments
12. SN74LS164, datasheet, Texas Instruments, http://www.ti.com
13. SN74166, datasheet, Texas Instruments
14. PIC16F87x, datasheet, Microchip Technology Inc, 2001

5 Intreruperi şi alte şmecherii hardware

5.1 In sfârşit, despre întreruperi

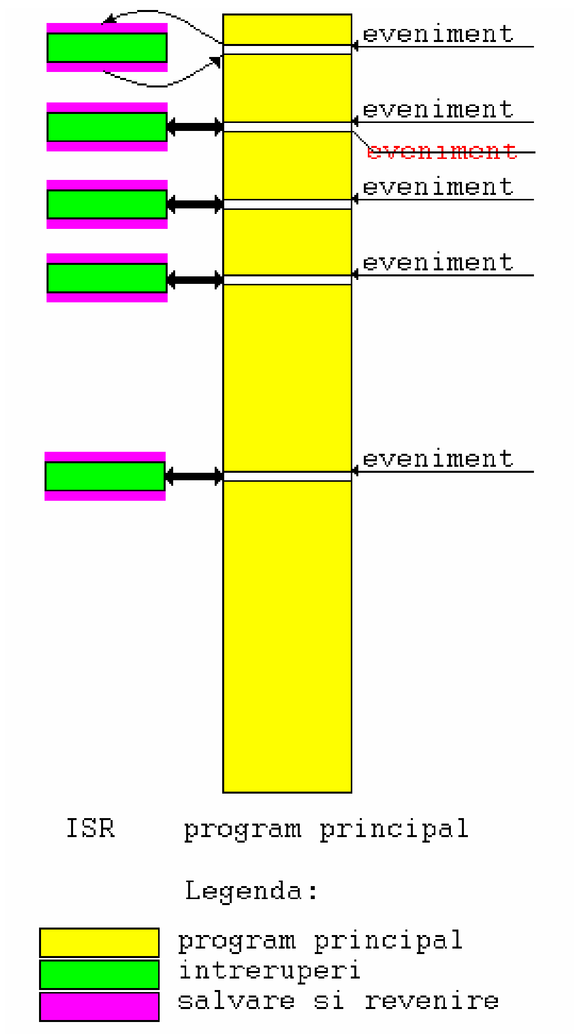


fig.5- 1 Tratarea unei întreruperi multiple

Intreruperile au o importanţă decisivă în elaborarea unui program coerent şi performant. Deşi microcontrolerul efectuează o singură operaţie cu exteriorul la un moment dat, el poate să întrerupă programul principal, fie la momente de timp determinate de acţiunea unor circuite integrate interfaţate cu el, fie la momente dictate de resursele sale hardware, să execute o altă porţiune de program a cărei acţiune se derulează mult mai rapid şi să revină apoi în programul principal. Modul de tratare a unui eveniment în întreruperi este evidenţiat în fig.5-1. Declanşarea orcărui eveniment duce la intrarea în execuţie a rutinei ISR (*Interrupt Service Routine*), recunoscută de compilator datorită instrucţiunii *pragma interrupt* sau *pragma raw_interrupt*. In momentul în care se este întâlnită comanda *pragma_interrupt*, compilatorul execută automat o secvenţă de program (cunoscută celor familiari cu microcontrolerul Z80 ca *push-pop*). Dacă utilizatorul se încăpăţînează să lucreze numai cu instrucţiuni de asamblare sau utilizează comada *pragma raw_interrupt* în rutina ISR (chiar dacă aceasta este editată în totalitate în JAL), va trebui să scrie această secvenţă singur la începutul şi sfârşitul ISR.

Compilarea unei secvente tipice JAL în care se utilizează întreruperile:

```
include 16f84_4
include jpic
procedure whats_new_Stan is
 pragma interrupt
end procedure
```

şi inspectarea filei *assembler* generate de compilator ne conduce la următoarea observaţie:

```
ORG 0000
  goto    __main
 ORG 0004          ;Salvarea registrilor esentiali (PUSH)
  movwf   H'0C'           ; movwf temporary_w
  swapf   H'03',w         ; swapf  status,w
  clrf    H'03'           ; clrf  status
  movwf   H'0D'           ; movwf temporary_status
  movfw   H'0A'           ; movfw pclath
  movwf   H'0E'           ; movwf temporary_pclath
  clrf    H'0A'           ; clrf  pclath
  movfw   H'04'           ; movfw FSR
  movwf   H'0F'           ; movwf temporary_FSR

  goto    __interrupt
 ORG 000E
__interrupt: ; 000E ; interrupt user code
_7577__vector: ; 000E
p_7577_whats_new_stan: ; 000E
e_7577_whats_new_stan: ; 000E
```

După salvarea registrului W, a registrului STATUS şi a FSR în regiştrii temporari, utilizând instrucțiunea *swapf* deoarece aceasta nu modifică registrul STATUS în timpul salvării lui, urmează corpul propriuzis al rutinei ISR, unde utilizatorul poate trata unul sau mai multe evenimente care-l interesează. Dacă evenimentele se derulează cu viteză mare şi rutina de întreruperi nu ține seama de durata necesară efectiv pentru a efectua instrucțiunile (1uS pentru tact de 4MHz respectiv 0.20uS pentru tact de 20MHz) este posibil ca evenimente ce apar pe parcursul duratei necesare tratării evenimentelor anterioare să nu poată fi procesate, fie din vina utilizatorului fie din cauză ca evenimentele se succed mult prea repede, astfel că acestea vor fi "pierdute". Un foarte bun exemplu este tentativa de citire a unui semnal cu factor de umplere foarte mic (raportul între perioadele când semnalul este în stare logică *high* respectiv în stare logică *low*, contorizată pe parcursul unei întregi perioade a semnalului). Dacă pulsul semnalului are durata comparabilă cu durata unui ciclu maşină şi procesorul nu are interfață hardware de comparare/captură, în mod cert o întrerupere pe RB0/int cu acest semnal nu va fi sesizată. Soluția este creşterea duratei impulsului şi simetrizarea acestuia prin utilizarea unui bistabil extern. Bistabilul va executa o divizare cu doi fiind necesară o corecție a algoritmului din ISR. Incheierea rutinei ISR se face cu secvența *pop*, care este o secventa *push* cu ordine inversată, iar întoarcerea în programul principal se face cu instrucțiunea *retfie*:

```
                      ; RESTAURAREA REGISTRILOR INITIALI (POP)
  movfw   H'0F'                  ; movfw temporary_FSR
  movwf   H'04'                  ; movwf FSR
  movfw   H'0E'                  ; movfw temporary_pclath
  movwf   H'0A'                  ; movwf pclath
  swapf   H'0D',w                ; swapf temporary_status, w
  movwf   H'03'                  ; movwf status
  swapf   H'0C',f                ; swapf temporary_w, f
  swapf   H'0C',w                ; swapf temporary_w, w
  retfie                         ; retfie
__main: ; 0017
```

Este important de notat că salvarea registrului *pclath* nu este necesară decât pentru microcontrolere ce au mai mult de 2kocteți de memorie program. Registrul **P**rogram

Counter are dimensiunea de 13 biţi fiind împărţit în **P**rogram **C**ounter **L**ow şi **P**rogram **C**outer **H**igh. Din aceştia numai PCL poate fi scris şi citit. PCH poate fi scris doar prin registrul PCLATH. Orice instrucţiune *call* sau *goto* generează 11 biţi de adresă, ceea ce este suficient pentru navigarea în pagini de memorie sub 2K. Biţii superiori sunt generaţi de PCLATH (bitul 4 şi bitul 3). Aceştia doi sunt cei vinovaţi de adresarea spaţiului de memorare superior graniţei de de 2K.

Retfie va seta automat bitul corespunzător din registrul INTCON responsabil cu întreruperile globale, (INTCON_GIE cap.3, fig.3-4) astfel că întreruperile vor rămâne active după interpretarea primului eveniment. Acest lucru obligă utilizatorul ca prima activare a întreruperilor să fie făcută în programul principal, altfel tratarea întreruperilor nu va avea loc. Rămâne la latitudinea aceluiaşi utilizator dezactivarea întreruperilor în momente cheie ale execuţiei programului principal utilizând de exemplu o procedură de dezactivare globală a întreruperilor (se pare că procedura de verificare a bitului INTCON_GIE după resetarea lui era necesară doar pentru primele microcontrolere Microchip, bug-ul respectiv fiind corectat în noile microcontrolere flash) sau numai dezactivarea întreruperilor specifice din regiştrii PIE şi INTCON, dacă anumite întreruperi trebuiesc să rămână active pentru "curgerea corectă" a programului principal.

```
procedure no_int is
 assembler
   local loop
  loop: bcf   intcon_gie      -- dezactivează toate intreruperile
        btfsc intcon_gie      -- fi sigur că au fost dezactivate
        goto  loop
 end assembler
end procedure
                         -- programul principal:
intcon_gie = high   -- întreruperi globale active
intcon_rbie = high  -- întreruperi la schimbarea stării active
intcon_t0ie = high  -- întreruperi ale TMR0 active
tmr1ie = high

forever loop
 ...
 no_int  -- dezactivează toate întreruperile
  -- linii utilizator, de exemplu: citeşte convertorul AD, compară rezultatul cu o constantă de 16 biţi,
dacă rezultatul este mai mic decât… salt la punctul A, dacă rezultatul este mai mare decât …atunci
salt la punctul B; deoarece întreruperile afectează citirea convertorului AD, saltul poate fi fals!
intcon_gie = high -- activează întreruperile globale
 ...
end loop
```

Numărul de întreruperi pentru seria flash midrange este variabil în funcţie de resursele interne ale fiecărui microcontroler şi poate fi maximum 15 (PIC16F877A) după cum urmează:

- **întreruperi externe** pe pinul RB0/INT şi intreruperi ale TMR0, flagurile corespunzătoare se găsesc în registrul INTCON (INTCON_INTE, INTCON_T0IE)
- **întreruperile perifericelor** sunt conţinute în regiştrii speciali PIR1 şi PIR2. Regiştrii de setare corespunzători sunt conţinuţi în regiştrii PIE1 şi PIE2 iar registrul global de setare a întreruperilor periferice în registrul INTCON (INTCON_PEIE)

Definirea exactă a tipului de intrerupere (internă sau a perifericelor) pentru PIC16F87x este prezentată în fig 5-2. Diferenţe semnificative se întâlnesc la:

- ❑ PIC16F8x care nu are decât 4 surse de întreruperi: TMR0, RB0/INT, schimbarea stării portului B, (Rb4…Rb7) şi scrierea completă a EEPROM – ului,
- ❑ PIC16F62x care are întreruperi identice cu PIC16F8x şi întreruperi specifice la modificarea stării comparatorului (bitul CMIE corespunzător registrului PIE1 şi INTCON_PEIE trebuie să fie setate pentru ca întreruperea să aibă loc) respectiv bitul CMIF corespunător registrului PIR1 va fi *high* dacă a avut loc o întrerupere prin comparator, CMIF trebuind să fie resetat software,
- ❑ PIC16F87xA care este o mixtură între PIC16F87x şi PIC 16F62x având atât întreruperile primului cât şi întreruperile specifice celui din urmă referitoare la comparatorul intern, are în total 15 surse de întreruperi

Semnificaţia biţilor din fig.5-2 este:

- T0IF (INTCON) este bitul de overflow al TMR0, T0IF = 1, a avut loc modificarea valorii registrului TMR0 de la FFh la 0h, resetarea software a T0IF este obligatorie (nu se resetează hardware). Numărarea începe de la valoarea existentă în TMR0; exemplu: pentru TMR0 = 254 şi prescalerul 1:1 sunt necesare doar două incrementări ale registrului până ce INTCON_T0IF îşi va schimba starea.
- INTF (INTCON) este bitul de întrerupere externă, INTF = 1 a avut loc o întrerupere externă pe pinul RB0/INT, resetarea software a INTF este obligatorie. Intreruperea RB0/INT este întreruperea principală pentru PIC16F fiind considerată cea mai sigură pentru identificarea evenimentelor externe microcontrolerului.
- RBIF (INTCON) bitul de întrerupere la schimbarea stării portului B<4…7>, RBIF = 1 înseamnă că unul din pinii Rb4…Rb7 şi-a schimbat starea, resetarea software este obligatorie. Această întrerupere poate fi dezactivată resetând INTCON_RBIE
- PSPIF (PIR1) bitul de intrerupere al portului paralel sclav, PSPIF = 1 înseamnă că a avut loc o operaţie de citire/scriere
- ADIF (PIR1) bitul de întrerupere al convertorului AD, ADIF = 1 înseamnă că s-a terminat o conversie AD
- RCIF (PIR1) bitul de întrerupere al USART, RCIF = 1 înseamnă că buferul Rx este plin
- TXIF (PIR1) bitul de întrerupere al USART, TXIF = 1, buferul Tx este gol
- SSPIF (PIR1) bitul de intrerupere al portului serial sincron, cu funcţii diferite în modul SPI şi I^2C
- CCP1IF (PIR1) bitul de întrerupere pentru modul compare/capture/pwm
- TMR2IF (PIR1) bitul de întrerupere la egalizarea valorilor PR2 şi TMR2
- TMR1IF (PIR1) bitul de overflow al TMR2, TMR1IF = 1 depăşirea a avut loc, resetarea software este obligatorie
- EEIF (PIR2) = 1, operaţia de scriere în eeprom s-a terminat, resetarea software este obligatorie
- BCLIF (PIR2) bitul de intrerupere la coliziunea pe bus-ul I^2C, BCLIF = 1, a avut loc o coliziune în mod stăpân
- CCP2IF (PIR2) bit de întreruperi pentru modul capture/compare referitor la TMR1

Tip PIC/întrerupere	T0IF (TMR0)	INTF (întrerupere externă)	RBIF/GPIF/RAIF (on change)	PSPIF (paralel sclav)	CMIF (comparator)	ADIF (convertor AD)	RCIF (USART)	TXIF (USART)	SSPIF (port serial)	CCP1IF (CCPWM1)	TMR2IF (TMR2)	TMR1IF (TMR1)	EEIF (EEprom)	BCLIF (port serial)	CCP2IF (CCPWM2)
PIC16F876/873	X	X	X	-	-	X	X	X	X	X	X	X	X	X	X
PIC16F874/877	X	X	X	X	-	X	X	X	X	X	X	X	X	X	X
PIC16F627/628	X	X	X		X	-	X	X	-	X	X	X	X	-	-
PIC16F83/84	X	X	X	-	-	-	-	-	-	-	-	-	X	-	-
PIC12F675/629	X	X	X	-	X	X	-	-	-	-	-	X	X	-	-
PIC16F630/676	X	X	X	-	X	X	-	-	-	-	-	X	X	-	-

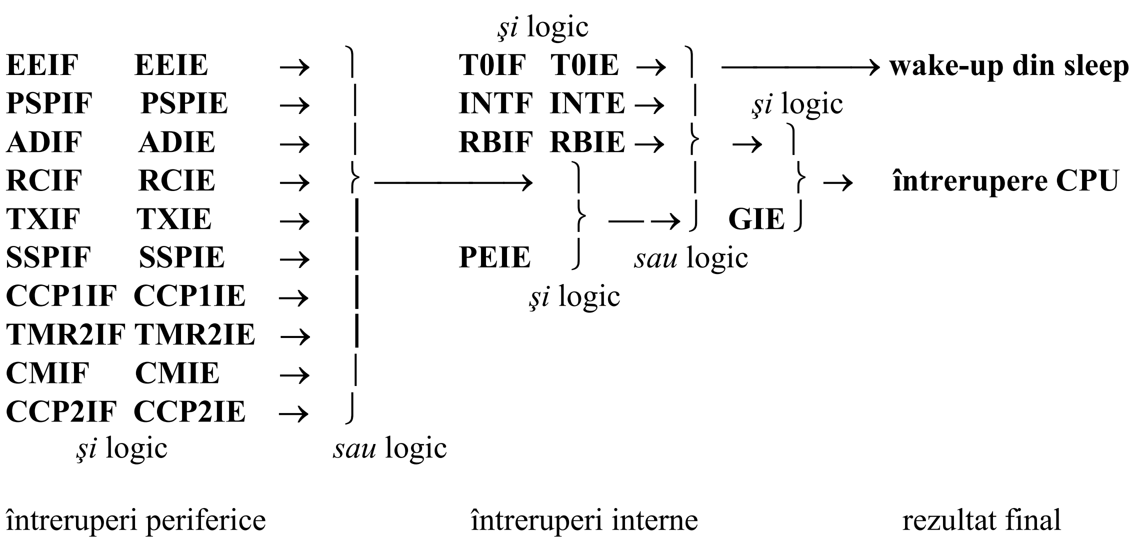


fig.5- 2 Sursele întreruperilor în PIC

Ideea de bază a funcționării tuturor întreruperilor este că întreruperea se poate activa sau dezactiva din regiştrii INTCON şi PIE (denumirea biților corespunzători au terminatia E) în timp ce flagurile care semnalizează că a avut loc o întrerupere anume, se găsesc în regiştrii INTCON şi PIR (denumirea biților corespunzatori au terminația F). Aceste flaguri trebuiesc resetate prin software după ce a avut loc evenimentul, pentru a da posibilitatea programului să sesizeze viitoarea întrerupere. In fig.5-2 întreruperile perifericelor sunt grupate în coloana din stânga, condiționările dintre flagurile E şi F fiind un *şi* logic iar acțiunea lor comună fiind de tip *sau*. Toate intreruperile periferice sunt condiționate de bitul PEIE prin *şi* logic, în timp ce bitul care condiționează toate întreruperile, inclusiv pe cele interne, este GIE.

PSPIF	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
7 R/W	6 R/W	5 R	4 R	3 R/W	2 R/W	1 R/W	0 R/W

PSPIF: flag de întreruperi pentru citire/scriere a portului paralel sclav
1 = o citire/scriere a avut loc, bitul se resetează software
0 = nu a avut loc citire/scriere

ADIF: flag-ul de întreruperi al convertorului AD
1 = o conversie AD s-a terminat
0 = conversia AD nu este completă

RCIF: flag de întrerupere pentru recepţia USART, nu poate fi scris doar citit
1 = buffer-ul USART de recepţie este plin
0 = buffer-ul USART de recepţie este gol

TXIF: flag de întrerupere pentru transmisia USART, nu poate fi scris doar citit
1 = buffer-ul USART de transmisie este plin
0 = buffer-ul USART de transmisie este gol

SSPIF: flagul de întrerupere al portului SSP
1 = condiţia de întrerupere a avut loc, resetarea software înainte de ieşirea din ISR este obligatorie. Poate fi setat de:

- SPI, dacă a avut loc o transmisie/recepţie
- I2C sclav, dacă a avut loc o transmisie/recepţie
- I2C stăpân, dacă:
  - a avut loc o transmisie/recepţie
  - comanda START, STOP sau RESTART a fost terminată de SSP
  - ACKnowledge-ul generat a fost terminat de SSP
  - un START sau STOP a fost generat cu modulul SSP inactiv (multimaster)

0 = nu a apărut nici o condiţie de întrerupere SSP

CCP1IF: flag-ul de întreruperi CCP1
Mod captură: 1 = a avut loc o captură prin TMR1, resetare software necesară
0 = nu a avut loc o captură prin TMR1
mod comparare: 1 = a avut loc o egalitate a registrului de comparare TMR1
0 = nu a avut loc o egalitate a registrului de comparare TMR1

TMR2IF: flag de întrerupere la coincidenţă TMR2/PR2
1 = a avut loc coincidenţa, resetare software necesară
0 = nu a avut loc coincidenţa

TMR1IF: flag de întrerupere la depăşirea valorii maxime a TMR1 (owerflow)
1 = registrul TMR1 a depăşit valoarea 255
0 = registrul TMR1 nu a depăsit valoarea maximă

PIR1

fig.5- 3 Registrul PIR1conţine flagurile pentru întreruperile externe

PSPIE	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE
7 R/W	6 R/W	5 R/W	4 R/W	3 R/W	2 R/W	1 R/W	0 R/W
PSPIE: flag pentru setarea întreruperii pentru citire/scriere a portului paralel sclav							
ADIE: flag pentru setarea întreruperii convertorului AD							
RCIE: flag pentru setarea întreruperii pentru recepţia USART							
TXIE: flag pentru setarea întreruperii pentru transmisia USART,							
SSPIE: flag pentru setarea întreruperii întrerupere al portului SSP							
CCP1IE: flag pentru setarea întreruperii CCP1							
TMR2IE: flag pentru setarea întreruperii la coincidenţă TMR2/PR2							
TMR1IF: flag pentru setarea întreruperii la depăşirea valorii maxime a TMR1 (overflow)							
Pentru toate flag-urile implicate: 1 = activează întreruperea respectivă 0 = dezactivează întreruperea respectivă PIE1							

fig.5- 4 Registrul PIE1 conţine biţii de setare individuală pentru întreruperi ale perifericelor

-	rezervat	-	EEIF	BCLIF	-	-	CCP2IF
7	6 R/W	5	4 R/W	3 R/W	2	1	0 R/W
Biţii neimplementaţi se citesc ca “0”							
Biţii rezervaţi se menţin resetaţi							
EEIF: flag de întrerupere pentru scriere în EEPROM 1 = scrierea a fost terminată 0 = scrierea nu este terminată sau nu a fost începută							
BCLIF: flag de întrerupere pentru coliziune pe bus 1 = a avut loc coliziune în SSP configurată pentru mod I2C master 0 = nu a avut loc nici o coliziune							
CCP2IF: întrerupere pentru activarea CCP2 Captura: 1 = a avut loc o captură de registru TMR1; resetare software necesară 0 = nu a vut loc captura TMR1 Comparare: 1 = a avut loc o coincidenţă a valorii registrului TMR1 0 = nu a avut loc coincidenţa PWM : nefolosit PIR2							

fig.5- 5 Registrul PIR2 conţine flagurile pentru întreruperi ale CCP2, SSP, EEPROM şi comparator

-	rezervat	-	EEIE	BCLIE	-	-	CCP2IE
7	6 R/W	5	4 R/W	3 R/W	2	1	0 R/W
Biţii neimplementaţi se citesc ca “0”							
Biţii rezervaţi se menţin resetaţi							
EEIE: întrerupere pentru activarea scrierii în EEPROM 1 = activează întreruperea 0 = dezactiveaza întreruperea							
BCLIE: întrerupere pentru activarea coliziunii pe bus 1 = activează întreruperea 0 = dezactivează întreruperea							
CCP2IE: întrerupere pentru activarea CCP2 1 = activează întreruperea 0 = dezactivează întreruperea PIE2							

fig.5- 6 Registrul PIE2 conţine biţii de setare individuală pentru întreruperile CCP2, coliziunile modulul SSP (port serial sincron), EEPROM şi comparator

5.1.1 Particularităţi ale întreruperilor în programele JAL

Fiecare compilator sau limbaj de programare are chichiţele lui. Nici Jal-ul nu stă mai prejos în ceea ce priveşte tratarea întreruperilor. Este esenţial modul în care utilizatorul implementează “împărţirea” timpului procesorului pentru tratarea programului principal (*main loop*) şi a rutinei ISR (*interrupt service routine*). Deşi nu mă pot declara un expert în întreruperi, există trei cazuri distincte pe care le-am observat în programele altor utilizatori sau le-am folosit (atât assembler cât şi limbaj de nivel înalt):

- Timpul de execuţie al programului principal este lung (peste 90% din timpul total de procesor), deservirea ISR este extrem de scurtă. Este situaţia din fig.5-1 şi poate fi considerat cazul ideal de funcţionare cu întreruperi. Evenimentele externe cu durate scurte sunt sesizate în proporţie de cel puţin 99%. Evenimentele generate de perifericele interne sunt sesizate în proporţie de 100%.
- Timpii de execuţie ai programului principal şi ISR sunt egali. Situaţia este posibilă pentru tratarea întreruperilor lente (ce se succed la nivelul sutelor de microsecunde). Nu poate fi folosită cu succes şi pentru întreruperi ale perifericelor interne, decât cu condiţia ca intervalul de repetare al evenimentelor dictate de acestea să fie suficient de mare.
- Timpul de execuţie al ISR devine 80%-90% din timpul total consumat de procesor în timp ce programului principal i se alocă restul. Este o situaţie anacronică care schimbă *main-loop*-ul cu ISR, dar care poate fi perfect funcţională dacă se folosesc un număr mic de întreruperi. Nu este implementabil pentru programe foarte lungi.

O precauţie aparte trebuie acordată utilizării anumitor instrucţiuni Jal când se urmăreşte tratarea rapidă a unei întreruperi, ca în exemplul următor:

```
-- setările pentru rutina de întreruperi
-- ------------------------------------
var bit apa is pin_b6
pin_b6_direction = input
var bit usa is pin_b7
pin_b7_direction = input
var byte w_temp
var byte status_temp
var byte fsr_temp
var bit apa_flag = low
var bit usa_flag = low
var volatile bit but1, but2, but3, but4
but1 = low but2 = low but3 = low but4 = low
var bit int_t0if = low
var byte kbd = 0
```

In această rutină de întreruperi se testează prin întreruperi, la schimbarea stării pinilor b6 şi b7, două semnale de intrare numite "apa" şi "uşa" care au prioritate maximă, urmate de citirea butoanelor but1…but4 şi generarea unui orologiu de timp real cu tmr1, pe intrarea de oscilator extern a acestuia fiind conectat un cuarţ de 32768 Hz. Secvenţele de *push* şi *pop* sunt marcate ca şi comentarii deaorece este prezentă comanda *pragma interrupt*. Dacă se utilizează *pragma raw_interrupt* ele trebuiesc inserate în program. In cazul detectării tranziţiei *low-high* pe oricare din intrările "uşa" sau "apa" se setează automat doi biţi, numiţi "uşa_p" respectiv "apa_p" care sunt utilizaţi convenabil în programul principal. Se poate observa că înaintea setării acestor biţi, se dezactivează întreruperile la schimbarea stării portului b, pentru a preîntâmpina detectarea altor tranziţii ce pot apare în timpul tratării întreruperilor ce deja sunt executate. Ieşirea din assembler se face cu activarea întreruperilor TMR1, pentru că următoarele instrucţiuni formează un ceas de timp real ce funcţionează prin decrementare.

```
    Procedure isr is
--------------------------------
  pragma interrupt
  assembler
    local  usa_p, apa_p, tmr
-- push
-- movwf  w_temp        -- salvare necesară pentru pragma raw_interrupt
-- swapf  status, w
-- movwf  status_temp
-- movf   fsr,w
-- movwf  fsr_temp
           btfss  intcon_rbif     -- testează dacă apare interrupt on change
           goto   tmr             -- dacă nu du-te la întreruperea tmr0
           btfsc  usa             -- dacă da, testează care pin
           goto   usa_p
           btfsc  apa
           goto   apa_p
           goto   tmr
    apa_p:
```

```
          bcf    intcon_rbie    -- dezactivează interrupt on change
          bsf    apa_flag
          goto   tmr
   usa_p:
          bcf    intcon_rbie
          bsf    usa_flag
   tmr:
          bsf status,5          -- bank_1
          bcf status,6
          bsf tmr1ie            -- activează întreruperile la depăşirea tmr1
          bcf status,5
          bcf status,6          -- bank_0
          bcf intcon_rbif       -- pregăteşte pentru o nouă interrupt on change
   end assembler

if tmr1if then second = second - 1 ; decrementează secunde
   if second == 255 then second = 59 minute = minute - 1 end if
   if minute == 255 then minute = 59 end if
end if

if intcon_t0if then kbd = kbd + 1 – întârziere pentru butoane
  if kbd == 3 then                   -- 3x65ms = 195ms
     int_t0if = intcon_t0if
     kbd = 0
  end if
end if
 if    (! Pin_a2) & int_t0if then but1 = on
 elsif (! Pin_a4) & int_t0if then but2 = on
 elsif (! Pin_a5) & int_t0if then but3 = on
 elsif (! Pin_e0) & int_t0if then but4 = on
end if
 int_t0if = low

   assembler
          bsf    intcon_rbie    -- setează pentru următoarea întrerupere
          bcf    intcon_t0if    -- curăţă t0if pentru următoarea întrerupere
          bcf    tmr1if         -- curăţa tmr1if
-- pop                          -- restaurează regiştrii
-- movf  fsr_temp, w
-- movwf fsr
-- swapf status_temp, w
-- movwf status
-- swapf w_temp, f
-- swapf w_temp, w
-- retfie
   end assembler
end procedure
```

Citirea butoanelor se face în întreruperi generate de TMR0 cu multiplicarea timpului maxim de *rollover* al acestuia de trei ori, folosind un numărător suplimentar numit *kbd*. Acesta permite inspectarea butoanelor cu o rată mai mare de timp necesară în aplicaţia respectivă. Detecţia butoanelor este însoţită de setarea variabilelor but1…but4 şi resetarea INTCON_T0IF. Flagurile but1…but4 sunt resetate în programul principal după ce sunt

utilizate pentru diferite comenzi. Ieşirea din ISR se face obligatoriu cu setări care să permită apariţia următoarelor întreruperi. De asemenea programul principal începe cu setări similare necesare detectării primelor întreruperi şi întrării corespunzătoare în ISR:

```
intcon_gie = high  -- activează întreruperile globale
intcon_rbie = high -- activează interrupts on change
intcon_t0ie = high -- activează întreruperile tmr0
-- main program
…
```

Se poate observa că rutina ISR prezentată, face parte din prima categorie despre care vorbeam. Practic întreruperile exemplificate pot fi considerate ca pseudoîntreruperi deoarece (înafara orologiului de timp) duratele desfăşurării lor depind şi de programul principal fiindcă aici se interpretează variabilele a căror setare are loc în ISR. Astfel, chiar dacă evenimentul a fost detectat, tratarea lui trebuie să se facă cu o asemenea viteză încât să nu se piardă următorul eveniment "apa", "uşa" sau "but1…but4"; ceea ce nu este cazul în aplicaţia de faţă, viteza de succedare a acestora fiind relativ mică (de ordinul milisecundelor). Rutina a fost utilizată în aceeaşi instalaţie de tratamente în câmp de microunde de mare putere prezentate în cap.3.1.2 Precauţiile ce trebuiesc luate în programul principal pentru tipul de ISR de mai sus sunt:

- O buclă *while…loop…end loop* nu este afectată de întreruperi dacă orice variabilă setată în ISR nu este citită şi resetată în interiorul instrucţiunii *while…loop* respective. Intreruperile care au loc în timpul execuţiei unei astfel de instrucţiuni *while…loop,* vor fi active după ce instrucţiunea şi-a epuizat de executat bucla.
- O instrucţiune *if…then…else* este afectată de întreruperi instantaneu cu apariţia acestora, excepţie fac utilizarea instrucţiunilor ce introduc întârzieri prin iterare ca *for…loop…end loop* în interiorul unei instrucţiuni *if…then…else,* întreruperea va fi activă doar la sfârşitul execuţiei acestui ciclu.
- Nu este recomandată apelarea aceloraşi rutine în ISR şi main, mai ales a operatorilor matematici predefiniţi.
- Se recomandă utilizarea la maxim a assemblerului pentru scrierea ISR deşi sunt permise *call*-uri ale altor rutine jal sau assembler.

5.2 Comanda triacelor cu microcontroler la trecerea prin zero a reţelei.

Una din problemele pe care utilizatorul de microcontrolere le întâlneşte atunci când este nevoit să comande tiristoare sau triace este comanda acestora la trecerea prin zero a reţelei de alimentare. Avantajele metodei sunt descrise în numeroase cărţi de specialitate [10], ideea principală este că se evită generarea unor zgomote perturbatoare în reţea, mai ales când sarcina este inductivă, importantă ca valoare (de exemplu, primarul unui transformator de 1KW/220V) şi triacul este protejat la apariţia vârfurilor nedorite de tensiune. Problema se rezumă la detectarea precisă a trecerii prin zero a reţelei şi aprinderea instantanee a triacului. Deasemenea poate fi necesară comanda PWM a sarcinii, acest lucru implică, pe lângă generarea precisă a momentelor de timp pentru comenzile *on-off* şi

sincronizarea comenzii *on* cu trecerea prin zero a reţelei. Pentru sarcini inductive este destul de dificilă sesizarea trecerii prin zero a curentului de sarcină (care este decalat în urma tensiunii şi necesită utilizarea unui transformator de curent), motiv pentru care în exemplul următor, comanda *off* nu este sincronizată cu reţeaua. Trecerea prin zero este sesizată prin intermediul unui triger schmitd de tipul MMC4093, de către RB0/INT. Este necesară prezenţa decodorului IC1 din două motive:

- Pentru comanda a 10 triace (nu sunt figurate decât 4) se utilizează doar 4 pini ai microcontrolerului.
- Separarea comenzii printr-un decodor binar-zecimal împiedică distrugerea triacelor în situaţia unei comenzi concomitente pe mai mult de o grilă, situaţie ce poate apărea în faza de alimentare tranzitorie a microcontrolerului, dacă comanda triacelor se face direct din microcontroler.

Se observă că nu există nici un circuit de izolare galvanică între microcontroler şi reţea, masa circuitului fiind conectată la pământ, respectiv la nulul reţelei. Interfaţarea acestui sistem cu un PC, trebuie făcută obligatoriu cu izolarea galvanică a comunicaţiei.

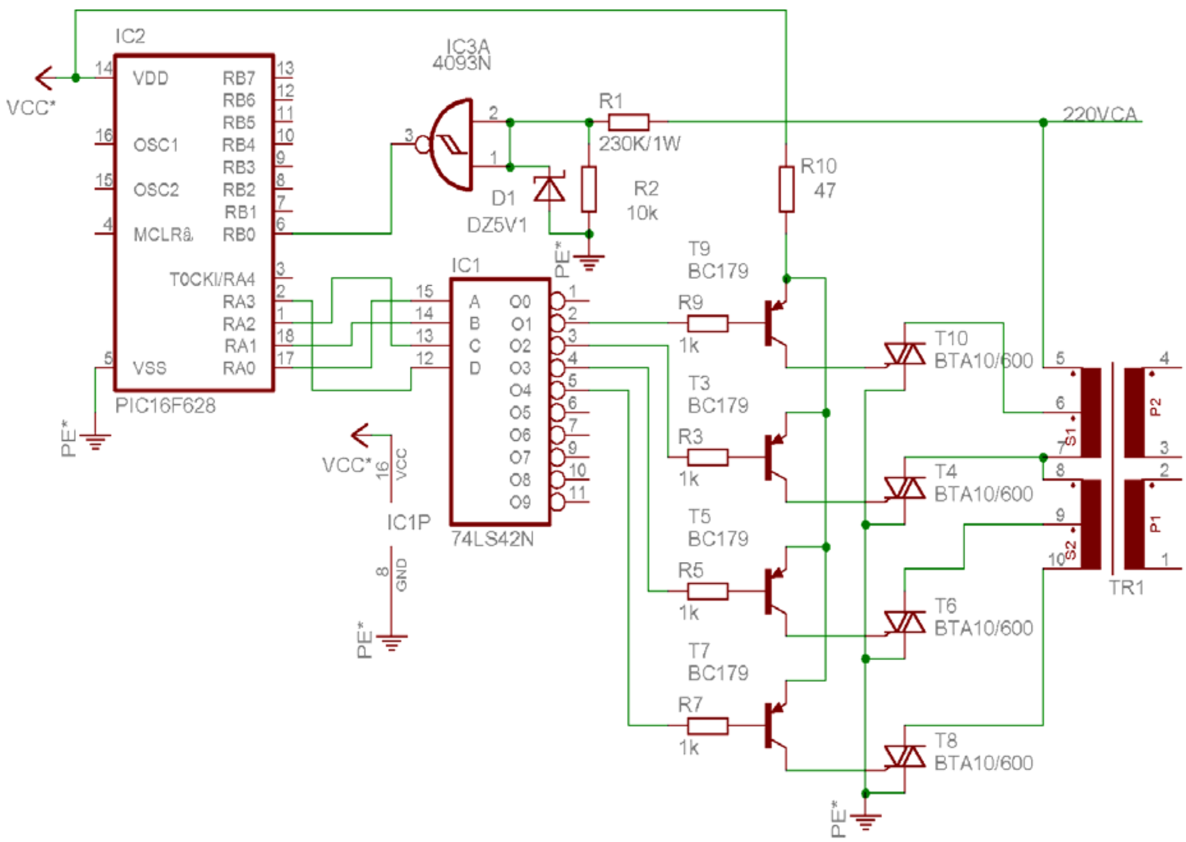


fig.5- 7 Interfaţarea triacelor cu comandă în cadranul III

Iată şi o porţiune din programul de comandă:

```
include 16f628_4
include jpic628
include digestp
include ds1820hi
include hd447804
include jprint
include jascii

pragma target fuses 0b_0011_1111_0001_0000
;  cp off, lvp off, boden off, mclr internal, pwrte on, intrc_io, wdt off
cmcon = 0x_07                                  ; fără comparatoare
var byte pwr  = 1
var bit start_prog = low
var volatile byte data is port_a_low ;  comanda pe 4 biți a ieşirilor
port_a_low_direction = all_output
var byte puls_c

var bit write = low
var bit filament is pin_a4
pin_a4_direction = output
filament = high

clear_watchdog              --  option este aici o pseudovariabilă !
option = 0b_0100_1000 ;   enable pullup port_b, prescaler asignat la wdt,
                      ;   int/rb0 front crescător
tmr0 = 0

procedure isr is
-- notă: această procedură este destinată instrucțiunii pragma  interrupt,
-- fără apelări din programul principal
pragma interrupt
if intcon_intf then          ;  eveniment la fiecare 100 milisecunde
   asm bcf intcon_inte       ;  dezactivează întreruperile externe rb0/int
 if start_prog then
   if puls_c < 10 then       ;  pentru un PWM < 100 %, dependent de valoarea lui puls_c
    if milisecond == 10 then data = pwr  end if
                      ;  pwr se modifică în rutina de programare parametrii, nu apare aici
    if milisecond == (10 - puls_c) then  data = 0 end if
                      ;terminare puls PWM
   end if
  if puls_c == 10 then data = pwr end if
                      ;  pentru acțiune continuă, PWM = 100%
 end if
end if
  asm bsf intcon_inte        ;  activează întreruperile externe
;  … alte secvențe de întreruperi
end procedure

asm bsf intcon_gie          --  activează toate întreruperile
bank_1
asm bsf tmr1ie              --  activează întreruperile tmr1 overflow
```

```
bank_0
uart_init                          --  inițializare USART

forever loop
; ****************
; …                              ;  alte rutine utilizator
 start_prog = on                 ;  flag de condiționare a unei secvențe din ISR
 write = on                      ;  flag de condiționare a scrierii în eeprom
 display_tmp_value ;  procedură de afişare a modului "măsură temperatură"
 temp_measurement                ;  rutină de măsură a temperaturii cu DS1820
     if ( minute == 0 ) & ( second == 0 ) then ;  timpul s-a scurs
          filament = high   ;  comenzi de ieşire
          start_prog = off   ;  dezactivare secvență în întrerupere, vezi ISR
           intcon_t0ie = low         ;  dezactivează  întreruperile tmr0 owerflow
          data = 0                   ;  scrie un nibble = 0 la ieşire
     end if
; …                                  ;  alte rutine utilizator
end loop
```

Deoarece este posibil ca exemplul să fie ceva mai greu inteligibil, fiind o mică parte dintr-un program ce comandă printre altele şi triace, sunt necesare câteva explicații suplimentare:

- "data" este o comandă pe 4 biți cu valoare cuprinsă între 0 şi 9 (BCD) care, prin intermediul unui decodor de tip 7442 acționează asupra comenzii de grilă a 10 triace, indiferent de valoarea pe care o are "data" doar un singur triac va fi comandat la un moment dat.
- "filament" este comanda de grilă a unui triac suplimentar ce alimentează un transformator de mică putere de cca. 50VA . Toți triacii sunt comandați în cadranul III.
- "puls_c" ia orice valoare de la 1 la 10 prin programare de la tastatură şi reprezintă valoarea de comparare care va dicta factorul de umplere al PWM-ului generat (de la 10% la 100%)
- rutina de întrerupere este completată cu algoritmul de generare al unei perioade precise de timp prezentate în cap.3.4.2 de această dată setările fiind făcute pentru 100mS şi deoarece algoritmul se repetă, nu a mai fost prezentat aici.
- o porțiune din ISR este validată sau nu prin intermediul unui flag "start_prog" comandat din programul principal.
- microcontrolerul utilizează oscilatorul şi reset-ul intern aşa cum setările lui *pragma target fuses* o definesc.

In exemplul anterior rezoluția PWM-ului ce comandă variabila de ieşire "data" a fost de 100mS. Acest lucru înseamnă un semnal activ de 0.1 S urmat de 0.9S pauză pentru o comandă memorată de 10%. In această situație prin semnal activ se înțelege un tren de impulsuri active fie pe nivel logic *low* fie pe nivel logic *high*, pauza având polaritatea opusă pulsului. Sunt situații în care e nevoie de o rezoluție mai slabă; de exemplu iată cum se poate genera un PWM cu o rezoluție de 1S (o comandă de 10% înseamnă 1S semnal activ urmată de 9S pauză):

```
procedure isr is
 pragma interrupt
;aici este amplasată una din rutinele de generare a secundei despre care am discutat în 3.4.2
if intcon_intf then
   asm bcf intcon_inte ; dezactivează întreruperile pe rb0/int
 if start_prog then
  if puls_c < 10 then
   if(second ==  0) | (second == 50) | (second == 40) |
     (second == 30) | (second == 20) | (second == 10)
   then data = pwr end if
-- generează comanda la fiecare 10 secunde
   if (second == 60 - puls_c) | (second == 50 - puls_c) |
      (second == 40 - puls_c) | (second == 30 - puls_c) |
      (second == 20 - puls_c) | (second == 10 - puls_c)
   then data = 0 end if
-- opreşte comanda la expirarea timpului dat de rezoluția PWM-ului (puls_c)
 end if

  if puls_c == 10 then data = pwr  end if
-- comandă validă permanent reprezentând un PWM de 100%
end if
end procedure
```

Cititorul îşi poate imagina cu uşurință că se poate genera orice PWM prin software având o rezoluție de cel puțin 50…100 de ori mai mare decât tactul procesor. Pentru durate mai scurte este nevoie de mult meşteşug în manipularea instrucțiunilor de asamblare sau utilizarea modulului CCP.

5.3 Dimensionarea corectă a unei surse de alimentare liniare pentru microcontrolere



Dealungul timpului, urmărind discuțiile ce au loc în grupul http://www.piclist.com susținut de un inimos administrator de rețea de la M.I.T. (James Michael Newton) am constatat cu stupoare că pentru cei mai mulți interlocutori (de obicei studenți) problemele principale în elaborarea unui produs *embedded technology* sunt cele legate de hardware şi nu de firmware. Cauza principală este cu siguranță şi lipsa experienței în abordarea corectă a proiectării surselor de alimentare, acestea fiind vinovate de obicei pentru toate dezastrele ce au loc în sisteme cu microcontrolere. Din fericire în ultimii 20 ani, în țară au apărut cărți valoroase ce tratează acest subiect. [1] [2]. De aceea voi puncta doar metodologia particulară de realizare a unei surse liniare de +5V destinată alimentării microcontrolerului. Datele de intrare necesare sunt:

- Puterea necesară la ieşire sau tensiunea şi curentul necesar de la sursă
- Variația tensiunii de alimentare (rețeaua de curent alternativ) la intrarea în sursă
- Regimul de lucru preconizat (funcționare continuă sau intermitentă). Pentru funcționare continuă se recomandă supradimensionarea radiatorului cu 30% din valoarea de calcul teoretic. Această supradimensionare ține cont şi de tipul de răcire utilizat (forțată sau naturală) respectiv de materialul din care este confecționată carcasa sursei

In situația particulară prezentată, vom alege tensiunea de ieşire Vo = +4.75V…+5.25V (standard de alimentare TTL), curentul necesar Io = 500mA, alimentarea se va face din rețeaua de curent alternativ 220V (+10% -15% adică 184V…242V) 50Hz. Din start, variația mare a tensiunii de alimentare (standardizată!) trebuie să dea de gândit asupra modalității de efectuare a calculelor: tensiunea minimă necesară pe circuitul integrat stabilizator se va calcula pentru valoarea minimă a rețelei (184V) iar puterea disipată de orice componentă activă sau pasivă a sursei, pentru valoarea maximă a tensiunii rețelei (242V). Acest lucru va duce la o creştere a disipației termice globale în sursă şi chiar dacă se va întâmpla doar o dată pe an, ca tensiunea din rețea să scadă la valoarea minimă garantată, sau să crească la cea maximă, sursa noastră va trebui să funcționeze corect chiar şi atunci. Schema acestei surse e atât de simplă încât ideea (greşită) este că ea nu poate să nu funcționeze corect (fig.5-8). O eroare tipică poate apare la elaborarea circuitului imprimat, unde asigurarea căii

cele mai scurte pentru curentul ondulatoriu (preluat de condensatorul de filtraj) este esenţială (mai ales la curenţi tari).

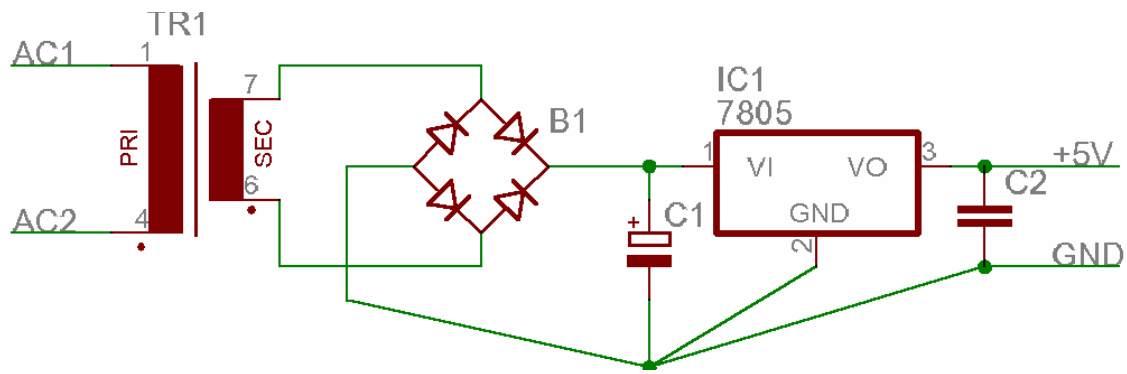


fig.5- 8 Cea mai simplă sursă de alimentare cu stabilizator monolitic cu 3 terminale

De aceea, ordinea de amplasare fizică pe circuitul imprimat este obligatoriu să fie: puntea redresoare B1, condensatorul de filtraj C1 şi apoi stabilizatorul monolitic IC1. Orice abatere de la acest traseu, ca de exemplu amplasarea condensatorului C1 la mare distanţă faţă de puntea B1 în timp ce IC1 se conectează direct la bornele punţii B1, poate duce la apariţia unor pulsuri datorate unor curenţi de masă paraziţi care acţionează asupra referinţei IC1, se regăsesc în ieşirea stabilizatorului şi nu pot fi suprimaţi nici dacă se măreşte cu două ordine de mărime condensatorul C2 de la valoarea normală de 100nF…10uF. De aceea este figurat modul de conectare al masei într-un singur punct cu impedanţa (calculată pentru c.a.) minimă, acesta fiind bornele condensatorul de filtraj. Atât referinţa stabilizatorului (pinul2) cât şi condensatorul C2 cu rol de suprimare al autooscilaţiilor parazite a stabilizatorului, trebuie să fie conectate cu traseul cel mai scurt la borna "-" a condensatorului de filtraj, a cărui pini vor fi deasemenea conectaţi cu traseul cel mai scurt spre puntea redresoare. A doua greşeală tipică este dimensionarea incorectă a condensatorului de filtraj. Amatorul va monta un condensator de filtraj "după ureche" probabil cât mai mare ca valoare pentru a avea factorul de ondulaţie minim pe intrarea stabilizatorului, fără să ţină seama că va "omorâ" în mod cert transformatorul. Osciloscoparea tensiunii alternative de intrare în puntea redresoare va evidenţia un semnal alternativ distorsionat, trapezoidal (în loc de unul curat, sinusoidal). Este semnul cert că, fie condensatorul de filtraj are valoare prea mare, fie transformatorul de alimentare nu poate asigura curentul de încărcare al condensatorului, fie puntea redresoare a suferit o străpungere parţială ireversibilă (situaţie valabilă numai la curenţi de peste 2A). Pentru a preântâmpina astfel de situaţii, un algoritm corect de calcul este următorul:

1. Se calculeaza tensiunea minimă necesară în intrarea 1 a stabilizatorului IC1. Conform datelor de catalog aceasta este: Vi = 4V + Vo = 9V pentru 7805. Se determină valoarea condensatorului C1. Acesta se încarcă în cca. 3mS şi se descarcă în 7 mS pentru o redresare bialternanţă (f = 100Hz, T = 10mS). Alegerea acestor valori de încărcare-descărcare sunt rezultatul observaţiei experimentale. Descărcarea condensatorului se realizează pe o rezistenţă echivalentă Rmin = Umin / Io, unde Io este curentul maxim absorbit de stabilizator (se poate neglija curentul de mers in gol al stabilizatorului). Pentru exemplul ales, Rmin = 9V/ 0.5A = 18 ohm.
2. Se calculează valoarea condensatorului impunând valoarea maximă admisă a riplului:

$$U\min = U\max \times e^{\frac{\Delta t}{R\min \times C1}}$$

Unde Δt = 7mS este timpul de descărcare al condensatorului pe rezistenţa echivalentă de sarcină, iar Umin = 9V, Rmin = 18ohm, Riplu = Umaix-Umin. Pentru un sistem digital valoarea maximă a riplului poate fi de ordinul 1…5V. Alegând Riplu = 4V rezultă Umax = 13V. Logaritmând expresia anterioară obţinem:

$$C1 = \frac{\Delta t}{R\min \times \left|\ln \frac{U\min}{U\max}\right|}$$

Pentru exemplul nostru C1calcul = 1057uF, se alege 1000uF. Căderea de tensiune pe puntea redresoare este de 2 x Vdioda = 2 x 0.6V = 1.2 V. Deci tensiunea alternativă efectivă a secundarului transformatorului trebuie să fie:

$$Uef = \frac{\sqrt{2}}{2} U\max + 1.2V$$

Adică pentru exemplul considerat, Uef = 0.707 x 13V +1.2V = 10.4V. Remarcaţi că această valoare este necesară pentru tensiunea de reţea minimă de 184V şi curentul nominal absorbit de 0.5A. Pentru 240V valoarea tensiunii va creşte cu 25% din valoarea anterioară :

Uef_{240} = 13.1V

3. Se calculează disipaţia maximă de putere pe stabilizator la 240V:
   $Umax_{240}$ = Uef_{240}/0.707 = 18.5V,această valoare determină şi tensiunea nominală a condensatorului C1 iar Umin = 11,3V
   Pdmax = (Umed – Vo)* Io = {($Umax_{240}$ + $Umin_{240}$)/2 – Vo}* Io = (15-5) * 0.5 = 5W unde V0=5V este tensiunea la ieşirea stabilizatorului iar Umed este căderea de tensiune medie pe stabilizator. Se observă că are loc o disipaţie importantă de putere pe stabilizator pentru o variaţie extremă a tensiunii de alimentare. Stabilizatorul ales trebuie să aibă puterea disipată de cel puţin două ori mai mare decât puterea calculată. Este evident că stabilizatorul necesită un radiator adecvat (7805 necesită radiator la puteri disipate mai mari de 2W şi nu poate funcţiona corect la mai mult de 6…8W - capsula TO220, chiar cu radiator). O soluţie rezonabilă este utilizarea unor stabilizatoare low-drop care necesită căderi de tensiune mult mai mici decat clasicul 7805, cuprinse între 0.5V şi 2.5V (seria LM29xx)
4. Se dimensionează puntea redresoare pentru o tensiune de lucru U > 2Umax (de regulă se alege puntea cu tensiunea minimă de lucru de 50V sau 100V) şi un curent I > (2…3)* Io

Observaţii: reducerea riplului tensiunii de intrare prin creşterea capacităţii condensatorului C1 va duce la scăderea amplitudinii tensiunii maxime necesare dar concomitent va duce la creşterea valorii curentului ondulatoriu prin punte, condensator şi transformator. Din această cauză producătorii de punţi redresoare recomandă utilizarea unei rezistenţe de limitare a curentului ondulatoriu, rezistenţă de valoare mică în serie cu capacitorul de

filtraj, recomandare care este uzual omisă la proiectare deoarece este un element disipativ în plus.

Concluzia ce se desprinde este că stabilizatorul liniar nu va putea fi folosit niciodată pentru obținerea curentului maxim precizat în foaia de catalog, datorită limitării date de puterea disipată a capsulei prin asigurarea diferenței minime de tensiune de intrare-ieşire necesară pentru funcționarea corectă (7805 are curentul maxim debitat în funcție de capsulă de: 100mA/TO72, 1A/TO220, respectiv 3A/TO3) Pentru curenți mai mari de 1-2A se recomandă realizarea unor surse de tensiune în comutație, cu precizarea că zgomotul de comutație indus de acestea în sistem este mai mare decât riplul la ieşirea unei surse liniare.

5.4 Flotarea microcontrolerului la tensiuni înalte

O idee preconcepută este aceea că modulul microcontroler trebuie să fie izolat galvanic de orice tensiune continuă sau alternativă a căror valoare depăşeşte cu mult tensiunea de alimentare a microcontrolerului. Dacă produsul realizat este cu funcționare independentă *(stand alone)* şi fără interfațări cu alte sisteme (sau cu interfațări izolate galvanic), este foarte posibil ca microcontrolerul să fie flotat peste o tensiune înaltă (de exemplu o tensiune continuă de +200V). Condiția imperativă de bună funcționare este dată de următoarele aspecte:

- Tensiunea înaltă nu trebuie să fie zgomotoasă
- Comenzile generate de microcontroler sunt disponibile față de masa acestuia şi nu față de masa sursei de înaltă tensiune.
- Este obligatorie realizarea îngrijită a cablajului imprimat pentru a evita curenți de scurgere capacitivi suficienți pentru a distruge porturile microcontrolerului.
- Eventualele tastaturi, butoane, LED-uri trebuie să fie bine izolate față de masa sistemului, masă conectată la pământ, pentru a asigura protecția operatorului şi fiabilitatea produsului.

O aplicație în care am utilizat acest principiu a fost un supraveghetor de flacără pentru un grup de 8 arzătoare cu gaz, utilizate într-un cuptor de uscare tunel. Deoarece singura metodă acceptată pentru supravegherea acestui tip de arzător este tija de ionizare (detectorii fotovoltaici sensibili în UV fiind perturbați de radiația arzătoarelor adiacente şi a ambrazurii cuptorului), sunt necesare câteva cuvinte despre principiul de funcționare al acesteia. Polarizând o tijă confecționată din material inoxidabil situată în flacăra de gaz cu tensiune continuă înaltă, va apare un curent de ionizare între aceasta şi corpul metalic al arzătorului legat la masă (şi la masa de protecție, adică la pământare). Dependența curentului de ionizare de tensiunea de polarizare are o alură exponențială până la tensiunea de 180V…200V după care, creşterea valorii acesteia este nesemnificativă chiar dacă tensiunea de polarizare ajunge la 1000V.

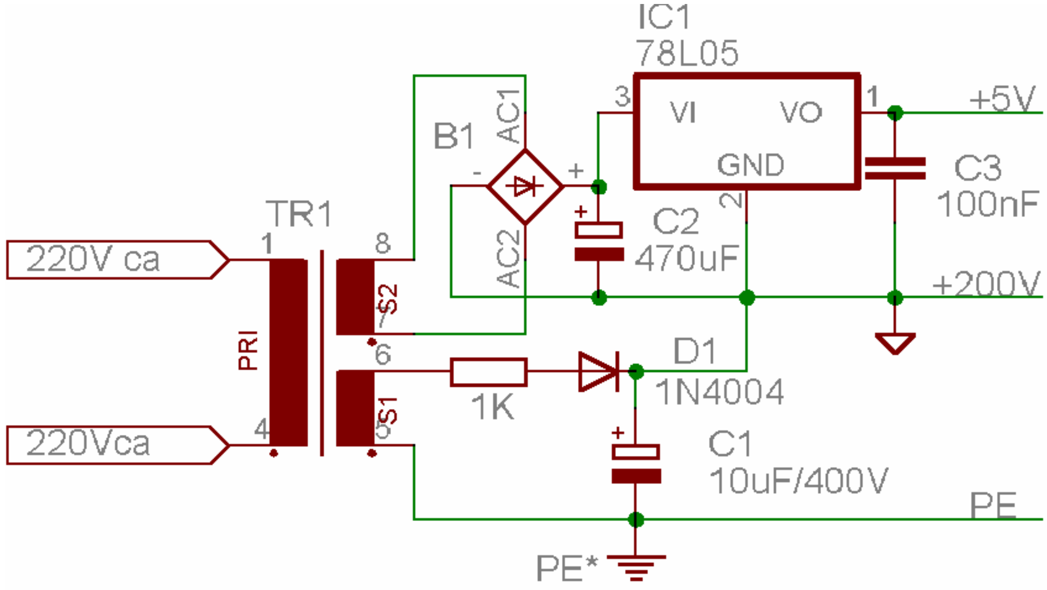


fig.5- 9 Alimentare flotantă "călărită" la +200V

Există o întreagă teorie şi metodologie privitoare la supravegherea flăcării de gaz pe care nu am intenţia să o detailez aici. Regula de bază este utilizarea redundanţei sistemului de supraveghere împreună cu îmbunătăţirea fiabilităţii designului, acesta din urmă tinzând spre perfecţiune. Este evident că obţinerea unui sistem electronic de supraveghere perfect este absolut imposibilă.

Sunt disponibile două surse de alimentare (fig.5-9): +200V (măsurată faţă de masa de protecţie PE) la un curent mic (limitarea acestuia fiind realizată de R1), tensiune ce generează curentul de ionizare şi o sursă de +5V pentru alimentarea microcontrolerului (măsurată faţa de aceeaşi masă flotantă +200V) sau +205V măsurată faţă de masa de protecţie PE. După cum se poate observa, microcontrolerul funcţionează flotant "agăţat" la +200V tensiune nestabilizatăcu riplu zero, deoarece curentul de ionizare are valoarea totală mult sub 1 mA pentru toate cele 8 supraveghetoare alimentate din acestă sursă. In fig.5-10, R1 este resistenţa de sesizare a curentului de ionizare şi trebuie să asigure potenţialul V_{GS}off = -1…-5V. R2 respectiv D1 asigură protecţia tranzistorului T1de tip FET-N, în situaţia unui scurt circuit accidental al tijei de ionizare (situaţie ce poate să apară la manevrarea defectuoasă a acesteia în faza de montaj). C1 este extrem de important şi trebuie calibrat în funcţie de frecvanţa de pâlpâire a flăcării, care este o constantă de arzător. Curentul de ionizare apare odată cu flacăra şi are o "curgere" convenţional teoretică dinspre +200V spre masa de protecţie PE. Lipsa curentului va deschide tranzistorul T1 până la R_{DS}on, prezenţa curentului îl va bloca. Semnalul preluat din drena T1 este interfaţat direct cu microcontrolerul, deşi portul B poate fi conectat cu rezistenţe interne de pull-up s-a preferat utilizarea rezistenţelor externe pentru o uşoară corecţie a dispersiei parametrilor interni ai tranzistoarelor FET. Microcontrolerul transmite datele culese de la tijele de ionizare ca un cuvânt de 8 biţi prin intermediul unei conexiuni industriale prin interfaţă RS485 optoizolată, la o unitate centrală ce semnalizează funcţionarea a 10 grupuri identice cu cel prezentat aici. *Modul1* până la *modul8* sunt copii ale detectorului curentului de ionizare grupat în jurul tranzistorului T1.

Precauţiile care se iau în situaţia exemplificată sunt:

- Facilitatea ICSP nu poate fi folosită decât dacă se deconectează tensiunea înaltă de pe modulul ce urmează a fi programat
- Măsurarea accidentală a tensiunilor pe pinii microcontrolerului față de masa de protecție cu un instrument cu impedanță mică de intrare, sau scurtcircuitarea accidentală a acestora chiar prin condensatoare, poate distruge microcontrolerul.
- Cutia în care se amplasează modulul electronic trebuie să fie din material izolator iar circuitul masei de protecție pe cablaj să fie bine separat

Intregul proiect poate să funcționeze cu alimentare clasică dacă se utilizează suplimentar încă 8 optocuplori în fiecare modul de supraveghere (modul1…modul7), pentru separarea tensiunii înalte de alimentarea microcontrolerului.

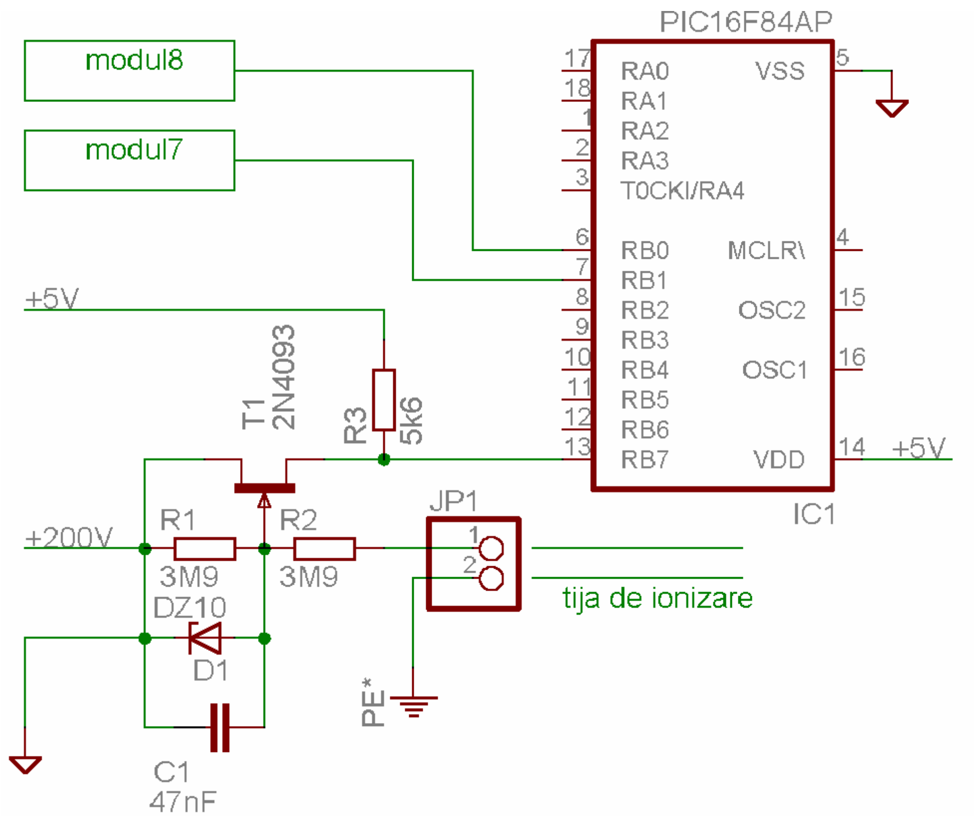


fig.5- 10 Interfațarea a 8 detectoare a curentului de ionizare la PIC

5.5 Alegerea tipului adecvat de oscilator extern

Microcontrolerele PIC funcționează cu patru tipuri de oscilatoare:

- Oscilator extern pe bază cristal de cuarț (modul LP până la 200KHz, XT: 200KHz până la 4MHz şi HS: 4MHz până la 20MHz)
- Oscilator extern pe bază de rezonator ceramic cu două terminale (condensatorii externi necesari)
- Oscilator extern pe baza de rezonator ceramic cu trei terminale (condensatori incluşi)
- Oscilator extern sau intern cu circuit RC (unele necesită doar rezistență exterioară)
- Oscilator extern independent

Pentru a nu vă întreba la nesfârşit de ce un cuarț nou nu oscilează când este conectat corect la PIC trebuie să aveți în vedere câteva aspecte:

- Microcontrolerele dispun de un cuvânt de configurare al "fuzibilelor" (capitolul *Special features of the CPU* al documentatiei microcontrolelelor) care trebuie setat corespunzator cu tipul de oscilator care este folosit. In jal acest lucru se face prin *pragma target_chip* sau *pragma config_fuses*. De remarcat că nu toate programatoarele pot seta aceste fuzibile din meniul propriu (de obicei din nişte butoane în fereastra windows), motiv pentru care această setare e bine să fie făcută în program. Astfel, la compilare, fila hexa va conține informația de configurare a fuzibilelor şi acestea vor marca automat *fuse*-urile în programator.
- Osciloscopul cu care se verifică funcționarea oscilatorului trebuie să aibă banda de frecvență ceva mai mare decât frecvența la care oscilează acesta. Cu un osciloscop modest cu banda de 10MHz se poate vedea de obicei oscilația unui cuarț de 10 sau 20MHz, dar amplitudinea acestuia va fi mai mică decât cea reală. Utilizarea unui osciloscop digital poate să dubleze semnele de întrebare pe care utilizatorul începător le are, deoarece un osciloscop având 20Msps (*megasample per second*) poate avea banda analogică de numai 5MHz sau chiar mai puțin. Deasemenea dacă impedanța de intrare a sondei este mai mică de 10Mohm, este posibil ca semnalul să fie atenuat de către sonda de măsură, în timpul măsurătorii.
- Pentru cuarțuri cu carcasă metalică nu strică să verificați în prealabil dacă nu este nici un scurtcircuit între fiecare dintre pinii activi şi carcasă
- Cuarțul (rezonatorul) trebuie conectat în imediata apropriere a microcontrolerului, în dreptul pinilor CLKIN şi CLKOUT şi nu la mare distanță de aceştia. Condiția este valabilă şi pentru condensatorii de amorsare ai oscilației. Singurul oscilator care poate fi montat pe placă, la distanță ceva mai mare de microcontroler este oscilatorul cu alimentare independentă care are ieşirea buffer-ată (amplificată) prin inversor logic.

Şi în fine, consumul unui microcontroler este dependent de frecvența la care acesta lucrează, limitele de consum fiind pentru PIC16F84 de 50uA în mod 32KHz, LP, de 5mA pentru 4MHz, XT, RC, respectiv 13mA pentru 4MHz, HS, fără nici o sarcină suplimentară la pinii acestuia. Nu vă aşteptați la rezultate extrem de precise (mai ales pentru măsurarea timpului) dacă utilizați rezonatoare ceramice. Nici chiar oscilatoarele cu cuarț de 32768 Hz folosite pentru ceasuri nu au frecvența identică cu ceea ce este marcat pe cuarț, de aceea utilizarea unui condensator trimer pe CLKOUT este importantă pentru ajustarea hardware a tactului. Termostatarea cuarțului este o metodă de îmbunătățire a stabilității acestuia cu două fețe, pe de o parte alegerea eronată a temperaturii de menținere a termostatului poate duce la apariția unei variații extrem de mari a frecvenței de oscilație, pentru o variație extrem de mică a temperaturii, deoarece fiecare cuarț în parte este sensibil altfel la temperaturi situate în limitele 45…70C, pe de altă parte dacă cuarțul este selectat corespunzător se pot obține rezultate extrem de spectaculoase.

In cazul în care este necesară furnizarea unui tact comun mai multor PIC-uri, există două variante posibile:

- Utilizarea unui oscilator extern cu cuarț sau rezonator ceramic pentru PIC1 şi conectarea CLKOUT a PIC1 cu CLKIN a PIC2
- Utilizarea unui oscilator independent extern şi conectarea comună a celor doi pini CLKIN al PIC1 respectiv CLKIN al PIC2, la ieşirea oscilatorului. Precauțiile privind liniile foarte lungi trebuie luată şi aici (planul de masă este obligatoriu).

5.6 Elemente hardware importante pentru funcționarea corectă a PIC-ului

Seria PIC16Fx poate funcționa la tensiuni cuprinse între 2 şi 5.5V, cu mici variații în funcție de specificul fiecărui tip în parte. PIC16F62x are domeniul tensiunilor de alimentare cuprinse doar între 3V şi 5.5V în timp ce PIC16LF87xA funcționează începând de la 2V la frecvențe mai mici de 4 MHz şi numai de la 3V pentru frecvențe mai mari de 10MHz. De aceea utilizatorul trebuie să studieze cu atenție fila de catalog cu specificațiile electrice pentru fiecare tip de capsulă în parte şi să coreleze informația găsită acolo cu frecvența la care microcontrolerul este utilizat în aplicația respectivă.

O atenție deosebită trebuie acordată setării BOR (*brown out detect*). Setarea acestui bit al cuvântului de configurare al fuzibilelor pune automat în funcțiune circuitul de resetare automată internă pentru situația când tensiunea de alimentare scade sub 3.6V…4.4V. Această dispersie este dată de referința internă şi circuitul de comparare, valoarea tipică fiind 4V. Deci nu uitați să resetați BOR dacă alimentați PIC-ul sub 4.5V fiindcă altfel programul dvs. se va transforma într-un ciclu infinit de start-reset.

Deşi fila de catalog specifică curentul maxim debitat sau absorbit de un singur pin ca fiind 20mA, este cu desăvârşire interzis consumul sau debitarea acestei valori simultan de pe toți pinii IO ai microcontrolerului, deoarece puterea maximă disipată de capsulă (1W) poate fi uşor depăşită. Performanțele electrice specificate de producător sunt contradictorii; de exemplu toate nivelele logice garantate sunt date la curenți debitați maximi I_{OL}= 8.5mA (V_{OL} = 0.6V) respectiv la curenți absorbiți I_{OH} = -3mA (V_{OH} = VDD –0.7V) deşi limitele sugerate în notele de aplicație sunt mult mai mari. De asemenea tensiunea maximă a pinului open drenă RA4 este numai de 8.5V, deşi producătorul recunoaşte că nu este o valoare testată ci doar tipică. O atenție sporită trebuie acordată programării cu tensiune redusă LVP. Pentru acest lucru trebuie setat bitul LVP din cuvântul de configurare şi trebuie să aibă loc în prealabil o primă programare normală cu HVP (de obicei acest lucru are loc în fabrică). După aceea, pinul RB4 trebuie conectat obligatoriu la masă cu o rezistență de 5K6…10K. Atât timp cât se lucrează cu LVP acest pin nu mai poate fi folosit decât ca şi intrare. De notat că facilitatea LVP nu este disponibilă pentru PIC16F8x.

Resetarea este o altă noțiune destul de controversată pentru seria PIC. In mod normal, cu o proiectare adecvată a sursei de alimentare (parametrul în discuție este viteza de variație până la stabilizarea tensiunii generate), circuitul de reset se compune dintr-o rezistență de pull-up pe pinul MCLR (R = 10K) şi de un condensator C = 100nF conectat de pe acelaşi pin la masă. Descărcarea condensatorului la deconectarea sursei de alimentare, se

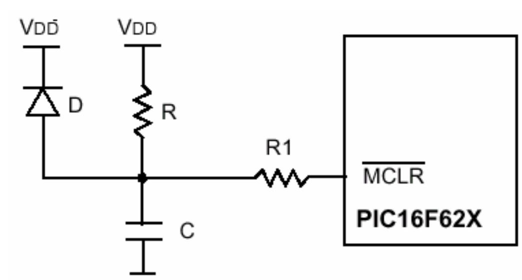


fig.5- 11 Circuitul tipic de reset sugerat de producător

poate face rapid cu ajutorul unei diode suplimentare conectate în paralel cu rezistența. Remarcați că dioda respectivă D, împiedică buna funcționare a ICSP (*in circuit serial*

programming). R1 are valoarea de 100 ohm până la 1Kohm şi are rolul de a proteja intrarea MCLR în cazul descărcării condensatorului prin intrare în urma unui stres ESD (***E**lectro**S**tatic **D**ischarge*). Experienţa arată că atât D cât şi R1 pot să lipsească fără nici o problemă de funcţionare, cu o sursă proiectată corect. Dacă viteza de variaţie a tensiunii de alimentare până la tensiunea nominală, este mai mare de 72 mS şi bitul PWRT (*power-up timer*) este resetat, va avea loc o întârziere în demararea programului vizibilă ca un reset prelungit rezultat numai ca acţiune a POR (*power on reset*) sau BOR (*brown out detect*). Adică, programul va începe numai după o întârziere cumulată egală cu timpul necesar stabilizării tensiunii de alimentare plus o întârziere fixă de 28…132mS (tipic 72mS). Deoarece timpul de stabilizare al sursei este de obicei sub valoarea de 72mS, doar dispersia valorii PWRT datorate variaţiei temperaturii cipului în limitele precizate de fila de catalog, pot să "dea peste cap" secvenţa de pornire. Dacă bitul PWRT este setat (bit conţinut în cuvântul de configurare al fuzibilelor), demararea programului are loc instantaneu cu alimentarea microcontrolerului, sursa trebuind să aibă un timp de stabilizare foarte scurt.

Bibliografie:

1. I.Ristea, C.A. Popescu - Stabilizatoare de tensiune, Editura tehnică, Bucureşti
2. Viorel Popescu – Stabilizatoare de tensiune în comutaţie, Editura de Vest, Timişoara, 1992

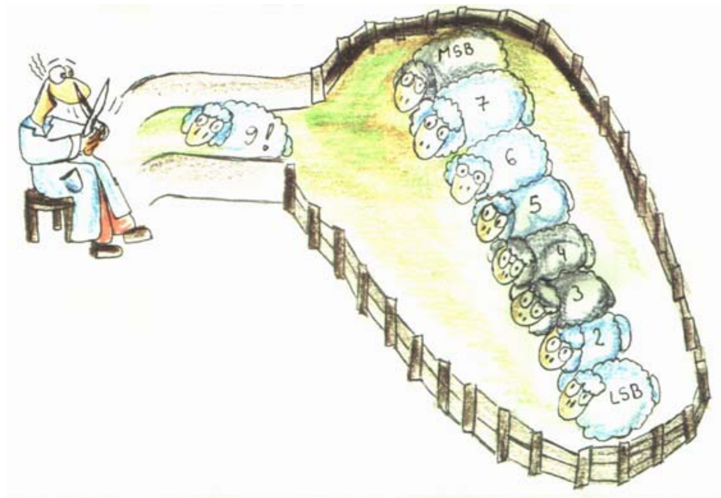
Note de aplicaţie din Embedeed Control Handbook, vol1, Microchip, 1997, DS0009D:

3. **AN585** – A Real-Time Operating systems for PIC micro Microcontrollers
4. **AN576** – Tehniques to Disable Global Interrupts
5. **AN566** – Using the PortB Interrupt on Change as an External Interrupt

File de catalog Microchip:

6. PIC16F8x - datasheet, DS30430C, 1998
7. PIC16F62x - datasheet, DS40300B DS80047B, 1999
8. PIC16F87x - datasheet, DS30292A DS30292B DS30292C, 1998-2001
9. PIC12F675 - datasheet, DS41190A, 2002
10. M.Bodea, şa – Diode şi tiristoare de putere, volII, Editura tehnică Bucureşti 1990

6 Comunicații seriale



Comunicațiile seriale reprezintă vârful metodelor de interfațare între sistemele ce conțin procesoare sau microcontrolere. Şi aceasta din trei motive: numărul mic de linii necesare (minim una, de regulă două sau trei), distanțele mari şi foarte mari ce pot fi acoperite, viteza de comunicație suficient de ridicată pentru aplicații comune. Am intrat deja în domeniu, în capitolul 3 abordând comunicația SPI prin metode software sau hardware. In acest capitol ordinea de zi conține cuvinte cheie ca: RS232 , I^2C, RS485.

6.1 Interfața RS232

Standardul RS232 este cu atât mai important pentru că permite interfațarea seriei PIC mid-range la calculatorul personal din orice generație, fie că este un biet PC din seria 486 / Pentium1 sau este vorba de un terminal miniatural de buzunar de genul Palmpilot. Standardul de conexiune RS232 este reprezentat de doi conectori rack cu 9 şi 25 de

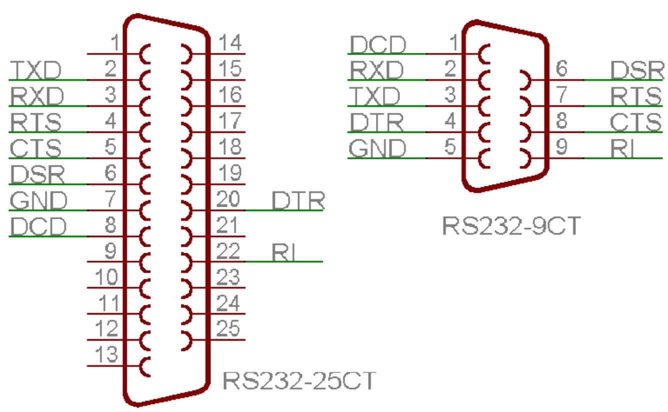


fig.6- 1 Conectorii standardizați pentru comunicația RS232

contacte, deşi există şi aparatură cu conectori superminiatură RS232 de tipul jack audio stereo (având doar RxD, TxD şi GND). In ceea ce priveşte modul de "curgere" al pachetelor de informație, sunt disponibile trei variante:

- Simplex, în care numai un echipament emite iar celălat receptionează
- Half-duplex în care pe rând fiecare echipament transmite în timp ce echipamentul opus recepționează
- Full-duplex în care simultan fiecare echipament transmite şi recepționează date

Semnificația pinilor conectorilor este cea menționată de abreviere:

```
TXD - ieşire transmisie date
RXD - intrare recepție date
GND - masă de semnal
RTS - Request To Send, ieşire de interogare a perifericului
DTR - Data Terminal Ready, ieşire de semnalizare pentru terminal
       liber
CTS - Clear To Send, intrare de acceptare a pachetului de date
DSR - Data Set Ready, intrare de validare a comunicației
DCD - Data Carier Detect, intrare de semnalizare a prezenței
       purtătoarei modemului
RI  - Ring Indicator, intrare de semnalizare a funcționării soneriei
       pentru terminalul opus
```

RTS şi CTS sunt folosite pentru a controla curgerea datelor dintre calculator şi periferic (modem). Când PC-ul este pregătit pentru a transmite datele, trece RTS în stare logică 1. Dacă perifericul este pregătit să recepționeze, va trece CTS în stare logică 1. Dacă PC-ul nu poate procesa datele dintr-un motiv oarecare, va dezactiva RTS trecându-l în 0 logic.

DTR şi DSR sunt utilizate numai pentru inițierea comunicației. Când PC–ul este pregătit pentru comunicația cu perifericul va seta DTR în nivel logic 1. Dacă perifericul este pregătit de recepție, va seta DSR pentru a informa PC-ul. Dacă există o eroare hardware în conexiune, perifericul va dezactiva DSR-ul informând astfel PC-ul despre problemă. Din semnalele menționate mai sus, trei sunt utilizate cel mai frecvent în comunicațiile actuale RXD, TXD şi GND, mai rar DTR, RTS, CTS şi DSR şi doar în interconectările cu modemuri DCD şi RI. Numărul mare de intrări-ieşiri este o reminiscență de acum 18…20 de ani când perifericele lente şi lipsa bufferelor de transmisie-recepție necesitau condiționări multiple ale comunicației prin semnale suplimentare. Cele câteva moduri tipice de conexiune între periferic (în cazul nostru microcontrolerul) şi PC sunt prezentate în fig.6-2.

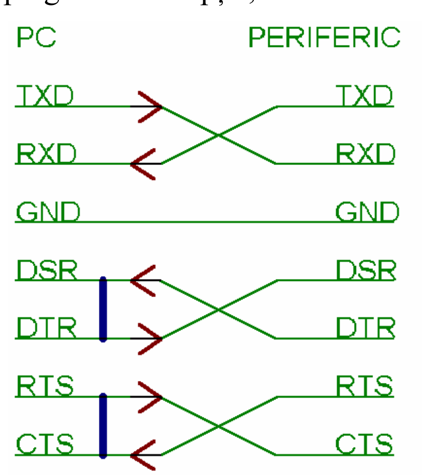


fig.6- 2 Conexiuni posibile în standardul RS232 între periferic şi PC

Se observă că numărul maxim de conexiuni este 7, însă acesta se poate reduce la 5 (RTS şi CTS lipsesc) sau la 3, (DSR şi DTR respectiv RTS şi CTS putînd fi conectați local) chiar şi pentru o comunicație full duplex. Ieşirile neutilizate pot fi folosite pentru a obține tensiune

de alimentare (de curent mic) necesară unor montaje electronice; metoda poartă denumirea de “furt de energie” din interfața serială. Comunicația acceptată de RS232 este asincronă spre deosebire de SPI sau I2C unde există un semnal de tact:

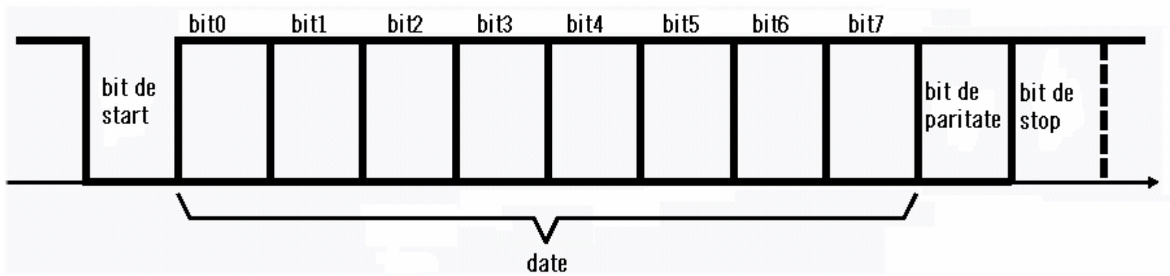


fig.6- 3 Formatul comunicației asincrone

De aceea pachetul trimis are nevoie de un bit de start pentru sincronizare, de la 4 la 8 biți de date ce transferă informația propriuzisă, un bit de paritate care poate fi dezactivat, par, impar, semn sau spațiu şi 1, 1.5 sau 2 biți de stop. Este agreat formatul cu 7 sau 8 biți de date, cele mai utilizate formate fiind 8N1 (8 biți de date, fără paritate, 1 bit de stop) sau 7N2. Bitul de paritate cu semn este reprezentat de bitul de paritate aflat întotdeauna în 1 logic; bitul de paritate cu spațiu reprezintă un bit aflat întotdeauna în stare logică 0; paritatea înseamnă că se verifică atât formatul biților de date cât şi al bitului de paritate, dacă ambele sunt 1 s-a transmis un cuvânt par, dacă ambele sunt 0 atunci s-a transmis un cuvânt impar. Bitul de stop are doar rolul de a da atât transmițătorului cât şi receptorului timp până la sosirea următorului pachet, de aceea pentru sisteme rapide, existența lui nu este foarte importantă. Bitul de paritate este în esență un decodor de eroare de 1 bit indicând dacă data a fost recepționată corect sau nu.

Viteza de comunicație reprezintă numărul de biți transmişi într-o secundă şi se măsoară în bps (*bit per second*) sau baud. Ratele de transmisie standardizate pentru RS232 sunt: 110, 300, 1200, 2400, 4800, 9600, 19200, (28800, 33600), 38400, 57600, 76800, 115200, 230400, 460800, 921600 bps. Rata maximă acceptată de PC este diferită de la generație la generație, în timp ce la 486 nu poate fi mai mult de 57600 bps, P1 acceptă 115200 bps iar P4 poate transmite chiar 921600bps. Adresele porturilor COM în PC în ordinea crescătoare a COM-urilor, sunt: 03F8, 02F8, 03E8, 02E8.

De notat că standardul EIA RS-232 nu permite conexiuni bidirecționale multipunct, există doar două echipamente pe linie ce comunică între ele full-duplex. Acest lucru nu înseamnă că nu se pot realiza comunicații simplex (TX şi GND), multiplexate în timp sau nu, între un terminal stăpân (*master*) şi mai multe terminale cu rol de sclavi (*slave*), utilizând aceeaşi linie de comunicație (sau un sistem radial de linii). Marginea de zgomot a semnalelor RS232 este mult mai bună comparativ cu standardul TTL (6V față de numai 2V pentru TTL, vezi fig.6-4), ceea ce explică distanțele mari acoperite (până la 900m). Remarcați că RS232 nu este foarte “standard” deoarece limitele superioare declarate ale valorilor tensiunilor pentru nivelul logic 1 pot fi de la -3V până la -15V, iar pentru nivelul logic 0 între +3V până la +15V pentru receptorul RS232, în timp ce emițătorul are nivelele cuprinse între ±5V…±15V. Tensiunile mici la nivelul emițătorului (maxim ±8V) sunt generate de obicei de unele *laptop*-uri, în timp ce interfețele cu destinație specială au tensiunile maxime de ±25V, fiind înafara precizărilor standardului, motiv pentru care, înainte de a interfața două echipamente pe RS232 este bine să verificați tensiunile generate

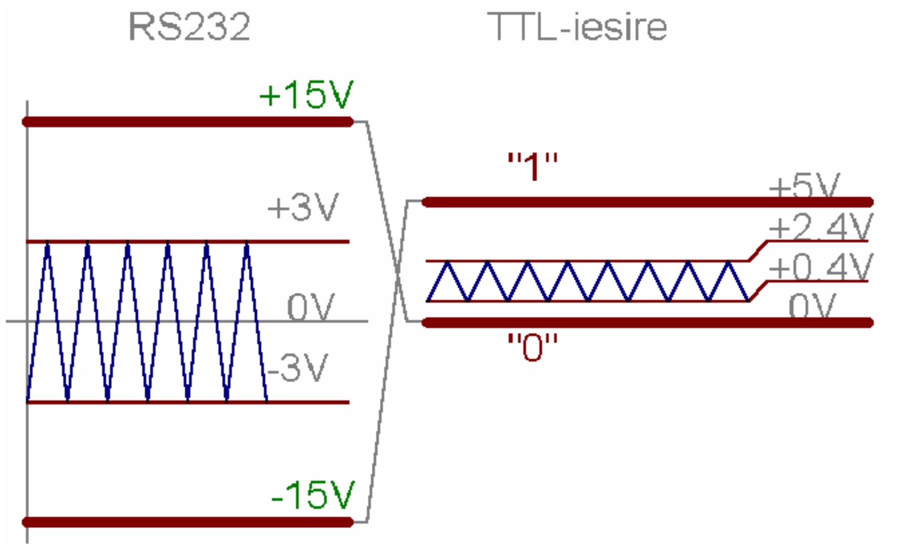


fig.6- 4 Nivelele logice ale RS232 comparativ cu semnalul TTL

de fiecare echipament terminal şi să înțelegeți de ce lungimea liniei poate fi doar 15m/19200bps pentru unele echipamente sau de 900m/2400bps pentru altele, în timp ce puterea disipată de receptor/emițător este diferită pentru diferse combinații de echipamente existente la capetele liniei, ducând uneori la încălzirea excesivă a convertoarelor de nivel. Impedanța de sarcină garantată a liniei trebuie să fie cuprinsă între 3…7 Kohm în paralel cu maxim 2500pF. Corelarea lungimii liniei cu tipul de cablu utilizat, viteza de comunicație şi impedanța de sarcină este un lucru absolut necesar. Distanțe foarte mari se pot obține doar cu cablu ecranat (cu dezavantajul unor capacități parazite mari) sau torsadat (capacități mai mici dar imunitate mai slabă la zgomote) unde fiecare semnal activ şi linia de masă fac perechi şi numai pentru viteze mici de comunicație ce nu depăşesc 4800 bps.

Cum se detectează pachetele de date care circulă pe RS232 este o altă poveste, uşor abordabilă împreună cu modul de interfațare la microcontroler.

6.1.1 Conversia PIC-RS232 utilizând rutine de tipul *busy-polling*

Noțiunea de citire prin *busy-polling* se referă la determinarea stării logice a unui pin de intrare în PIC, utilizat ca linie RxD, într-o buclă nesfârşită, realizată prin metode software ce au la bază obținerea unor întârzieri precise de timp. Este evident că tactul microcontrolerului care generează operațiunea de citire, trebuie să aibă frecvența mult mai mare decât viteza de comunicație, altfel citirea corectă a fiecărui bit din pachetul recepționat este imposibilă. Atât timp cât nu se detectează prezența caracterului ce urmează a fi recepționat, microcontrolerul nu face nimic altceva decât să aştepte într-un ciclu “*busy-looping*”sosirea acestuia. Dacă tactul procesor este suficient de ridicat, întârzierile pot fi înlocuite cu diverse operațiuni utile dar care au întotdeauna aceeaşi lungime, consumând întotdeauna aceeaşi cicli maşină (pentru a asigura întârzieri reproductibile). Cunoscând

viteza de comunicație a pachetului şi numărul de biți pe care acesta îl are, se poate afla cu certitudine intervalul de timp necesar pentru detectarea unui bit. Acesta este:
T_bit [μS] = 1 / baud_rate [bps]. El poate fi, pentru câteva rate de comunicație, următorul:

Viteza[bps]	timp[uS]
9600	104
19200	52
38400	26
76800	13

Pentru un PIC funcționând la 4 MHz, tactul procesor va fi exact de 1μS datorită existenței divizorului intern cu 4. Condiția ca citirea pachetului să fie corectă, este ca utilizatorul să asigure o întârziere de aproximativ jumătate din această perioadă după detectarea bitului de start, pentru ca fiecare detecție ce urmează, să prindă cu siguranță o stare stabilă a bitului şi nu regimul tranzitoriu ce poate avea loc la momentul tranziției dintre doi biți consecutivi purtând informație de polaritate opusă. Pentru exemplificare se consideră transmisia octetului 10101010:

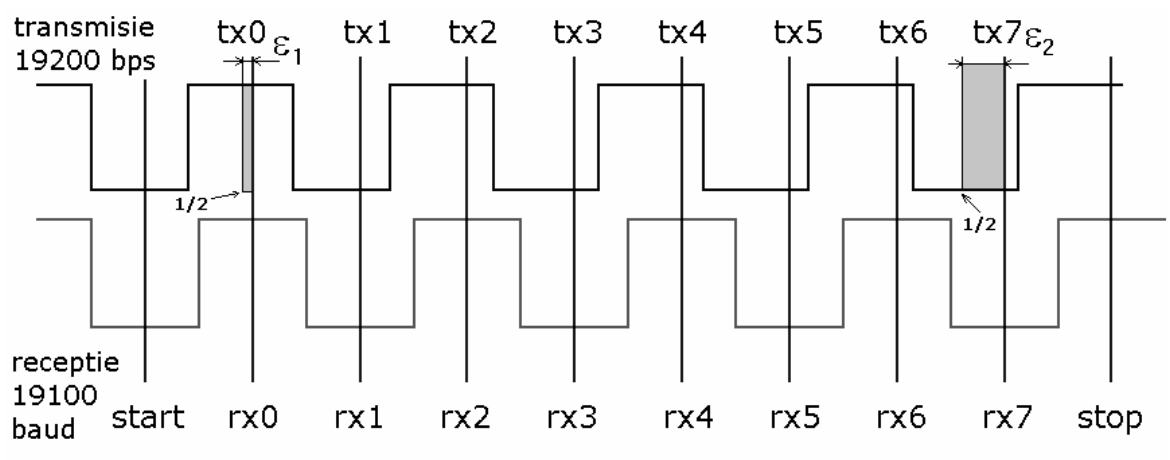


fig.6- 5 Eroarea de *sampling* între două semnale având viteze diferite de comunicație

Dacă sincronizarea se face simultan pe primul front căzător al semnalelor de transmisie respectiv de recepție (fig.6-5), odată cu bitul de start, primii biți recepționați (rx0…rx4) vor fi corecți chiar dacă momentul detecției s-a modificat vizibil față de mijlocul perioadei bitului transmis (referința este considerată a fi semnalul la recepție), în timp ce biții rx5…rx7 au şanse din ce în ce mai mari de a fi recepționați greşit pe măsura îndepărtării de momentul sincronizării ($\varepsilon_2 >> \varepsilon_1$). Eroarea este deci aditivă până în momentul unei noi sincronizări (recepția unui nou bit de start). Pentru refacerea datelor prin metode *busy-polling* se consideră acceptabilă o eroare de maxim 3% între vitezele de comunicație ale emițătorului şi receptorului. Exemplul din fig.6-5 (exagerat pentru evidențierea fenomenului) prezintă o eroare de 0.5% ceea ce este mai mult decât perfect pentru o comunicație reală. Această eroare se datorează imperfecțiunii oscilatoarelor cu cuarț sau cu rezonator, utilizate în PIC sau în PC. De remarcat faptul că rezonatoarele au frecvența mult mai dependentă de temperatura ambiantă decât cuarțurile iar precizia lor este mai proastă comparativ cu a cristalelor de cuarț .

Rutina ce execută operația de recepție/transmisie pentru viteza de 19200bps este următoarea:

```
-- jseriala; bibliotecă pentru comunicație busy-polling
procedure delay10 is -- 10 µS incluzând call-urile
                     -- call=2 + return=2 + 3*(goto=2)
  assembler
    local L1, L2, L3
        goto L1
    L1: goto L2
    L2: goto L3
    L3:
  end assembler
end procedure
procedure delay44 is -- 44 µS incluzând call-urile
                     -- call=2 + return=2 + 4*10
  delay10
  delay10
  delay10
  delay10
end procedure
```

Asigurarea întârzierilor necesare se face prin înlănțuirea unei rutine de întârziere de 10µS şi a uneia de 44µS rezultată tot din prima.

```
procedure asynch_send( byte in x ) is
   var bit current_bit at x : 0
   var byte times
   assembler
     local sendloop, L1, L2, by0, by1
                     -- aşteaptă doi biți ( 52+52=104µS )
     call delay44
     call delay44
     call delay10
     call delay10          -- 108 este acceptabil
     bcf asynch_out_pin
-- generarea bitului de start, începe numărătoarea întârzierii generate
     call  delay44         -- 44
     movlw   8             -- cei 8 biți de transmis,    1
     movwf   times         -- 1
                           -- 44+2=46
     goto    L1            -- 46+2( goto)=48
L1:  goto    L2            -- 48+2=50
L2:
sendloop:
     btfss   current_bit   -- asynch_out_pin = current_bit
     goto    by0
     bsf     asynch_out_pin
     goto    by1
by0:
     bcf     asynch_out_pin
     nop
by1:                            -- 5
     rrf     x, f               -- 6
```

```
     call    delay44          --  44+6=50
                              --  end loop
     decfsz times, f          --  51
     goto   sendloop          --  53,suficient de bine
                              --  generarea bitului de stop
     bsf   asynch_out_pin
     call  delay44
     call  delay10            --  55
  end assembler
end procedure

var byte last_received
procedure asynch_receive is
   var byte times
   assembler
      local WaitIdle, WaitStart, RecvLoop
                              --  aşteaptă pentru lipsă bit
     WaitIdle:
        btfss asynch_in_pin   --  activ low
        goto  WaitIdle
                              --  aşteaptă pentru bitul de start
     WaitStart:
        btfsc asynch_in_pin   --  activ low
        goto  WaitStart
                              --  aşteaptă aproximativ o jumătate de bit ( 26 )
        call    delay10
        call    delay10       --  20
        nop                   --  1
                              --  încarcă counterul pentru repetare de 8 ori:
        movlw   8             --  1
        movwf   times           --  1
        goto    recvloop        --  2+20+3=25, aproape bine
      recvloop:
        call    delay44         --  44
        nop                     --  1
        clrc                    --     1
        rrf   last_received, f  --     1
                                --  bit x:7 = asynch_in_pin
        btfsc  asynch_in_pin    --     1
        bsf    last_received, 7 --     1
                                --  bucla s-a terminat dacă s-a rotit al 8-lea bit
        decfsz times, f         --     1
        goto   recvloop         --     2
                                --  aşteaptă pentru primul bit de stop
        call    delay44         --  44+8=52, perfect!
   end assembler
end procedure
```

Acest tip de rutine se folosesc numai în microcontrolere ce nu dispun de modul USART incorporat, ca PIC16F8X sau 12FXXX, sau când e nevoie de mai mult de o transmisie serială (cazul multiplexorului de RS232 realizat cu microcontroler). Un exemplu simplu care utilizează un PIC16F84 (sau PIC16F628 setat fără USART) pe post de terminal,

repetând caracterele tastate sub programul Hyperterminal (existent în mediul Windows) pe un afişaj cu cristale lichide şi apoi trimițându-le înapoi spre PC este prezentat în continuare.

Noutatea intervenită în fig.6-6 este convertorul de nivel IC2, MAX232 (tensiunea minimă de alimentare +5V) sau MAX3232 (tensiunea minimă de alimentare +3V). De remarcat că se găsesc pe piață circuite integrate echivalente cu aceste cipuri (ST232CN) cu preț de cost mult redus şi funcționare identică. Programul terminal are doar câteva linii:

```
include 16f84_4  -- definirea fuzibilelor şi a tactului
include jpic     -- definirea regiştrilor PIC
include max232p  -- bibliotecă de configurare hardware
include jseriala -- conține rutinele seriale 19200,8,n,1
include hd447804 -- interfațarea la lcd în modul 4+2 fire

hd44780_clear            -- initializare lcd
hd44780_line1            -- cursor LCD pe linia 1 char0
var byte count = 0       -- contor

forever loop             -- execută la nesfârşit următoarele:
asynch_receive           -- recepționează un caracter pe pinul rx
 if count == 8 then      -- verifică dacă prima linie LCD e completă
  hd44780_line2          -- dacă da, sari pe linia 2 a LCD
 elsif count > 15 then   -- verifică dacă a doua linie e completă
  count = 0              -- resetează numărătorul
  hd44780_clear          -- şterge afişajul
  hd44780_line1          -- cursor LCD pe linia1 char0
 end if                  -- termină testările
 hd44780 = last_received -- scrie pe lcd
 count = count + 1       -- incrementează numărătorul
asynch_send ( last_received )
                      -- trimite înapoi la PC caracterul recepționat
end loop              -- şi reia de la capăt
```

Singura specificare suplimentară trebuie făcută pentru biblioteca max232p care configurează conexiunile hardware la LCD şi la convertorul de nivel MAX232:

```
-- fila              : max232p.jal
-- IMPORTANT : include hd44780p se marchează ca şi comentariu în  hd447804.jal, pentru a lăsa
-- active liniile de mai jos referitoare la HD44780

var volatile bit tx is pin_a0
pin_a0_direction = output
var volatile bit rx is pin_a1
pin_a1_direction = input

 var volatile bit  hd44780_4_DI is pin_b4
 var volatile bit  hd44780_4_E  is pin_b7
 var volatile bit  hd44780_4_RW is pin_b5
 var volatile byte hd44780_4_D  is port_b_low

 procedure _hd44780_4_init is
   port_b_low           = 0
   pin_b4               = low
```

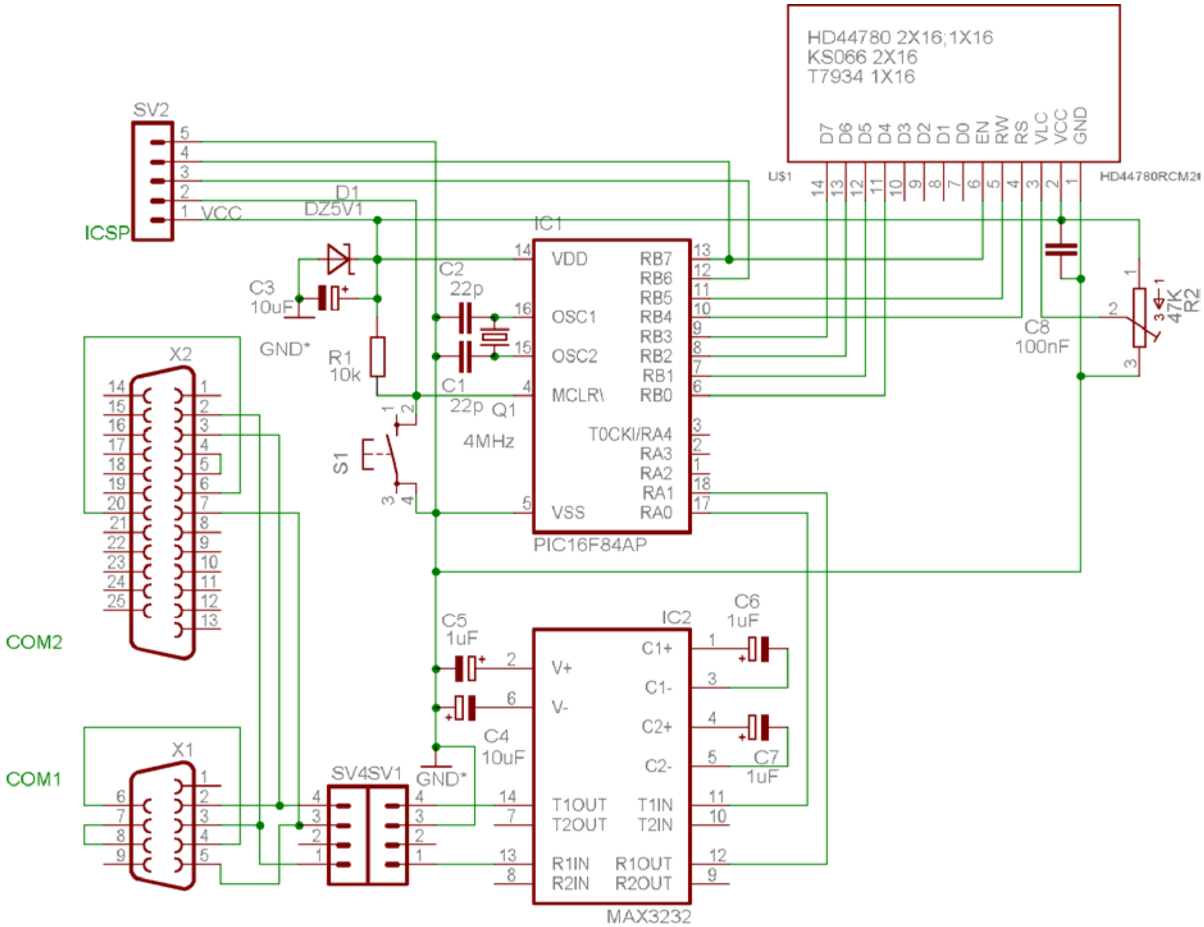


fig.6- 6 Un terminal cu PIC16F84 şi afişaj LCD compatibil HD44780

```
  pin_b5                = low
  pin_b7                = low
  port_b_low_direction  = all_output
  pin_b4_direction      = output
  pin_b5_direction      = output
  pin_b7_direction      = output
end procedure
```

MAX232 are în componența sa două inversoare, deci semnalele dinspre şi spre PC, vor fi întotdeauna inversate ca polaritate față de semnalele TTL existente la intrarea/ieşirea pinilor de comunicație ai PIC-ului, care sunt active *low*. Dacă se intenționează utilizarea unui convertor de nivel neinversor realizat cu componente discrete, polaritatea semnalului trebuie analizată în consecință. Avantajul utilizării circuitului MAX232 pentru conversia nivelului este simplitatea circuitului şi posibilitatea utilizării a două tensiuni de ± 8…10V existente pe pinii 2 şi 6, pentru alimentarea unor blocuri analogice ce pot fi necesare în aplicația utilizator. Altfel, este perfect posibilă utilizarea unui singur tranzistor PNP alimentat din interfața serială a PC-ului, ca schimbător de nivel pentru linia Tx a interfeței TTL-RS232, linia Rx necesitând o banală rezistență pentru conversia inversă RS232-TTL.

6.1.2 Conversia PIC-RS232 utilizând modulul USART

Dezavantajul major al comunicației *busy-polling* este modul destul de greoi al utilizării întreruperilor sau al programelor ramificate, deoarece intervalele de timp contorizate sunt stricte. Nici viteza de comunicație nu poate fi prea mare decât cu artificii importante la 4 MHz sau utilizând doar PIC-uri ce lucrează la 20MHz. De aceea apariția microcontrolelor flash cu USART încorporat şi preț de cost extrem de scăzut (PIC16F628) a fost extrem de benefică pentru utilizatori. USART-ul asigură transmisia asincronă full-duplex, cu rate de transmisie standardizate până la cca. 1Mbps pentru frecvențe de funcționare ale microcontrolerului de 20MHz. Poate funcționa deasemenea în mod sincron half-duplex ca master sau slave. Există cinci regiştrii importanți care guvernează funcționarea USART: TXSTA registru de setare a parametrilor transmisiei, RCSTA registru de setare a parametrilor recepției, registrul generator al vitezei de comunicație SPBRG şi regiştrii de transmisie TXREG şi de recepție RXREG a caracterului.

Registrul TXSTA permite selecția numărului de biți (8 sau 9 prin bitul TX9), modul de comunicație (sincron sau asincron prin bitul SYNC), selecția vitezei ridicate de comunicație (BRGH) şi începerea transmisiei (TXEN).

CSRC	TX9	TXEN	SYNC	-	BRGH	TRMT	TX9D
7 R/W	6 R/W	5 R/W	4 R/W	3	2 R/W	1 R	0 R/W
CSRC: bitul de selecție al sursei de tact Mod asincron: nu contează valoarea Mod sincron: 1 = mod master, tactul este generat intern din BRG 0 = mod sclav, tactul este generat din sursă externă							
TX9: bit de selecție pentru transmisia bitului 9 1 = selectează transmisia cu 9 biți 0 = selectează transmisia cu 8 biți							
TXEN: bitul de startare a transmisiei 1 = transmisia este activată 0 = transmisia este dezactivată SREN/CREN rescrie TXEN în modul SYNC							
SYNC: bitul de selecție al modului de funcționare USART 1 = mod sincron 0 = mod asincron							
BRGH: bitul de selecție al comunicației de viteză ridicată Mod asincron: 1 = viteză mărită 0 = viteză scăzută Mod sincron : neutilizat							
TRMT: bitul de status al registrului de rotire (Transmit Shift Register) 1 = TSR este gol 0 = TSR este plin							
TX9D: al nouălea bit al datei transmise, poate fi bit de paritate TXSTA							

tab.6- 7 Registrul de transmisie TXSTA

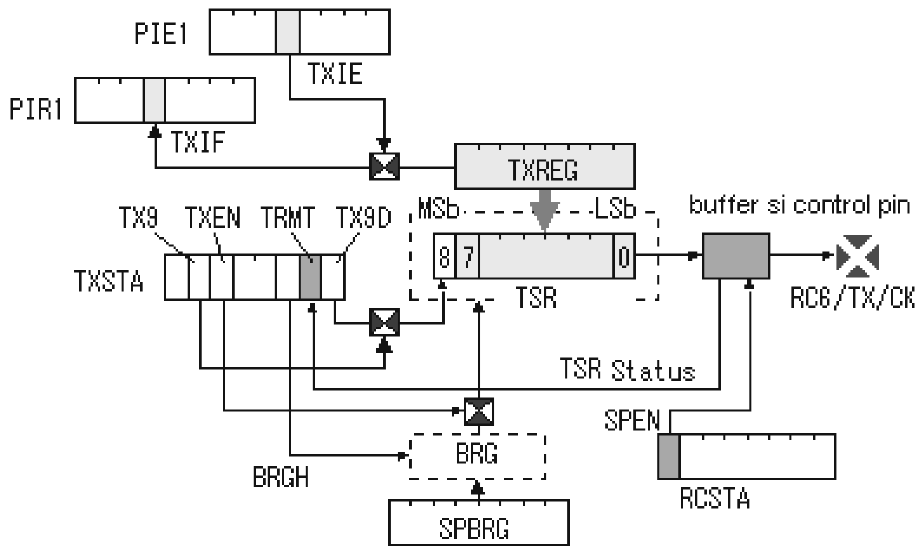


fig.6-7 Efectuarea transmisiei prin USART

La transmisie, SPEN trece în stare logică *high* iar portul RC6 devine ieşire TX. Data existentă în registrul TXREG este transferată hardware în **T**ransmit **S**hift **R**egister de unde este serializată (primul bit transmis este LSB) spre pinul de ieşire RC6. Bitul TXEN (registrul TXSTA) setat *on*, demarează transmisia. In acest moment TXIF (PIR1) devine *high*, şi are loc întreruperea. Atenție, TXIF nu poate fi resetat software! Această întrerupere semnalizează că a fost golit conținutul TXREG în TSR şi poate avea loc o nouă înscriere a acestuia. Transmisia poate fi fluentă numai dacă detecția acestei întreruperi este făcută în mod continuu. Bitul TRMT (TXSTA) devine *high* de fiecare dată când s-a golit TSR. Verificarea periodică a acestuia prin polling, confirmă golirea TSR. De remarcat că TSR nu poate fi scris sau citit direct. Viteza de golire a TSR este dictată de BRG. Paritatea nu este suportată prin hardware dar poate fi generată în mod software şi memorată ca al nouălea bit. In acest caz, bitul de paritate este transmis prin setarea lui TX9D *on* în registrul TXSTA, iar apoi setând bitul TX9 din acelaşi registru. TX9D trebuie setat înainte de a transmite data în registrul TXREG. Acest mod de transmisie startează imediat ce data a fost încărcată în TXREG. Dacă TX9D nu a fost setat în prealabil, are loc o transmisie normală fără bit de paritate. Dacă TXEN este resetat în timp ce comunicația are loc, aceasta încetează şi RC6 trece în impedanță ridicată. Paşii necesari la realizarea unei transmisii sunt:

- Inițializarea SPBRG pentru rata de transmisie dorită, setarea BRGH pentru viteză mărită dacă este cazul.
- Activarea portului asincron prin resetarea bitului SYNC al TXSTA şi setarea bitului SPEN al RCSTA.
- Dacă sunt necesare întreruperi, setarea bitului TXIE al registrului PIE.
- Dacă este necesară transmisie pe 9 biți, setarea bitului TX9 al registrului TXSTA.
- Activarea transmisiei prin setarea bitului TXEN al TXSTA, TXIF din PIR1 devine *high,* semnalizând posibilitatea scrierii în TXREG.
- Dacă a fost selectată transmisia pe 9 biți, bitul 9 trebuie încărcat în TX9D.
- Transmisia are loc în momentul încărcării datei de transmis în TXREG.

Registru de recepție RCSTA permite activarea sau inhibarea recepției (SPEN), selectarea numărului de biți recepționați la 8 sau 9 (RX9), activarea recepției în mod continuu (CREN) şi detecția a două tipuri de erori: *frame error* (FERR) când se recepționează caractere eronate datorită unei incompatibilități între viteza transmițatorului şi cea a USART-ului setată prin valoarea registrului SPBRG, respectiv *overrun error* (OERR), când s-a recepționat un nou caracter fără ca registrul de recepție RCREG să fie golit după a treia detecție de caracter (când *fifo*-ul intern de doi biți ce copiază valoarea registrului RCREG a fost depăşit).

SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D
7 R/W	6 R/W	5 R/W	4 R/W	3R/W	2 R	1 R	0 R

SPEN: bitul de setare al portului serial
1 = portul serial este activ (pinii RX şi TX sunt configurați ca pini ai portului serial)
0 = portul serial este dezactivat

RX9: bitul de selecție pentru recepția bitului 9
1 = selectează recepția bitului 9
0 = selectează recepția a 8 biți de date

SREN: bitul de selecție pentru recepția unui singur octet de date
Mod sincron, stăpân: 1 = setează recepția singulară
0 = dezactivează recepția singulară
Mod asincron şi sincron, sclav: nu contează

CREN: bitul de activare al recepției continue
Mod asincron: 1 = activează recepția continuă
0 = dezactivează recepția continuă
Mod sincron: 1 = setează recepția continuă până când CREN este resetat
0 = dezactivează recepția continuă

ADDEN: bitul de selecție pentru detectarea adresei
Mod asincron pe 9 biți (RX9=1) :
1 = activează detectarea adresei, setează întreruperile şi citeşte bufferul de recepție când RSR:8 este setat
0 = dezactivează detecția adresei, toți biții sunt recepționați, bitul 9 poate fi folosit ca bit de paritate

FERR: bitul de eroare la recepție fragmentată
1 = eroare de fragmentare, poate fi şters prin citirea RCREG şi receptia următorului octet valid
0 = nu este eroare de fragmentare

OERR: bitul de semnalizare al erorii prin depăşire
1 = a avut loc o eroare prin depăşire, se poate şterge resetând bitul CREN
0 = nu a fost eroare

RX9D: al 9-lea bit al datei recepționate (poate fi bit de paritate, calculat de utilizator)

RCSTA

tab.6- 8 Registrul de recepție RCSTA al USART

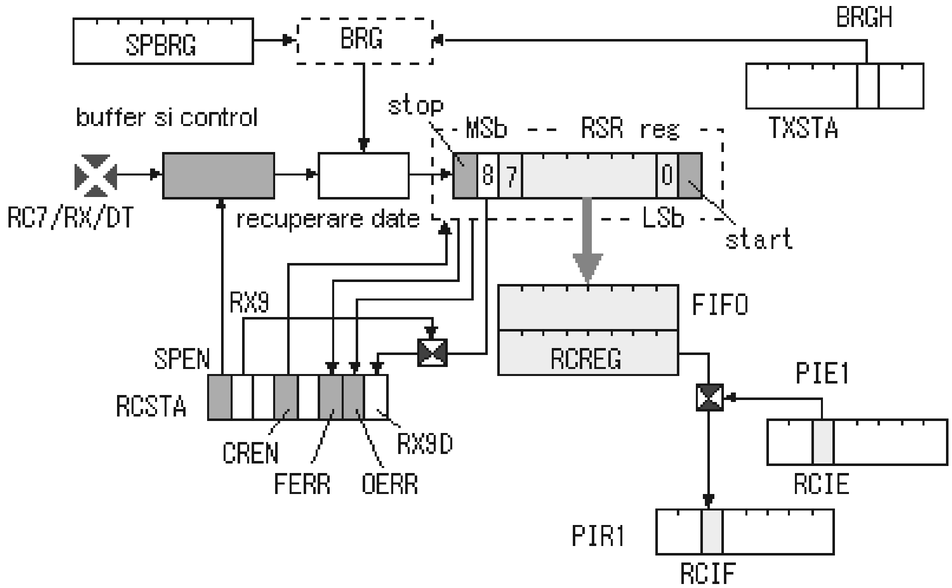


fig.6-8 Recepția prin USART

La recepție, bitul SPEN al RCSTA devine *high* şi transformă RC7 în pin de intrare. Data care a fost recepționată la portul RX ajunge în registrul RSR prin registrul de recuperare *data recovery.* Are loc o eşantionare a datei de intrare pe front căzător, repetată de trei ori, după care este memorată în RSR cu viteza specificată în registrul SPBRG respectiv şi de bitul BRGH al TXSTA. Când se detectează un bit de stop, conținutul registrului RSR este transferat în RCREG. Când data este memorată în RCREG, bitul RCIF al PIR1 devine *high*. Pentru a valida aceasta ca întrerupere, trebuie inițial setat bitul RCIE al PIE1. RCREG este alcătuit din 2 FIFO şi poate memora 2 octeți, ca o protecție pentru întârzieri software în procesul de citire. RCIF nu poate fi decât citit, fiind şters la citirea RCREG. Dacă RCREG nu a fost citit până la terminarea recepției în RSR, bitul OERR al RCSTA devine *high,* şi se semnalizează eroare. Data care a fost memorată în acest timp în RSR este pierdută. Operația de recepție nu se termină, bitul CREN al RCSTA şi OERR este resetat. Bitul FERR al RCSTA este setat când se detectează eroare de recepție în RSR. RX9D şi FERR sunt rescrise de fiecare dată când se recepționează un octet. Bitul FERR trebuie verificat înainte de a fi citit conținutul registrului RCREG. Când se recepționează o secvență corectă după una eronată, informația stocată în FERR dispare. Secvența necesară la recepție este următoarea:

- Inițializarea SPBRG şi/sau a BRGH pentru rata de comunicație corespunzătoare aplicației
- Setarea portului serial prin resetarea bitului SYNC al TXSTA şi setarea bitului SPEN al RCSTA
- Dacă sunt necesare întreruperi, setarea bitului RCIE al PIE1
- Dacă este necesară recepția pe 9 biți, setarea lui RX9 în RCSTA
- Activarea recepției prin setarea lui CREN în RCSTA
- Bitul RCIF al PIR1 devine *high* dacă recepția este completă iar întreruperea este generată prin setarea prealabilă a bitului RCIE din PIE1
- Citirea registrului RCSTA pentru obținerea bitului 9 şi determinarea apariției orcărui tip de eroare în timpul recepției

- Citirea datei recepționate din RCREG
- Stergerea orcărei erori apărute la recepție se face prin resetarea bitului CREN din RCSTA

Aşa cum am observat, viteza de comunicație a USART-ului cade în seama registrului SPBRG atât la transmisie cât şi la recepție. Formula ce stabileşte această viteză este diferită pentru modul de lucru cu viteză redusă (bitul BRGH = 0) sau ridicată (BRGH = 1) :

spbrg = (xtal[hz]/(baudrate[bps]*64))-1 pentru brgh = low
spbrg = (xtal[hz]/(baudrate[bps]*16))-1 pentru brgh = high

Valoarea spbrg poate fi de minimum 0 şi maximum 255, fiind un registru de 8 biți. Xtal este frecvența oscilatorului cu cuarț măsurată în Hz.
Limitele posibile pentru trei frecvențe rotunde de lucru sunt prezentate în tabelul următor:

xtal(MHz)	**baud_rate**	**spbrg/brgh**	**spbrg/brgh**
20	1200	255 0	
	2400	129 0	
	4800	64 0	
	9600		129 1
	19200		64 1
	28800		42 1
	33600		36 1
	38400		32 1
	57600		20 1
	115200		10 1
10	1200	129 1	
	2400	64 0	
	4800	31 0	
	9600		64 1
	19200		31 1
	28800		21 1
	33600		18 1
	38400		15 1
	57600		10 1
4	300	207 0	
	1200	51 0	
	2400	25 0	
	4800	12 0	
	9600		25 1
	19200		12 1

Se observă că valoarea registrului SPBRG nu poate fi decât întreagă. Utilizatorul poate să-şi pună pe bună dreptate întrebarea: care este eroarea cu care funcționează în mod real comunicația față de valoarea standardizată a vitezei de comunicație ? Această eroare este diferită pentru cele două moduri menționate anterior şi anume:

Eroarea = (baud_rate_calcul – baud_rate_real) * 100 / baud_rate_real

Unde:

Baud_rate_calcul = xtal[Hz]/(64(spbrg + 1)) pentru brgh = low respectiv
Baud_rate_calcul = xtal[Hz]/(16(spbrg + 1)) pentru brgh = high

reprezintă vitezele de comunicație calculate pentru valorile SPBRG rotunjite la întreg cu relațiile de calcul anterioare ce generează şi tabelul de mai sus, iar *baud_rate_real* este viteza de transmisie standardizată. Erorile vor fi mult diferite pentru diverse combinații SPBRG/BRGH şi valori ale cristalului de cuarț. O limită acceptabilă pentru care comunicația va funcționa în acest caz este de 5%. Pentru minimizarea erorii se pot utiliza cuarțuri speciale a căror frecvență este un multiplu de 1.8432Mhz (3.6864MHz, 7.3728MHz sau 14.7456MHz) construite special pentru generarea timingului în comunicațiile seriale. Calculând eroarea pentru un PIC lucrând la 20MHz, cu 115200, brgh = high, spbrg = 10 vom obține o viteză de comunicație reală, baud_rate_real = 113636bps ceea ce corespunde unei erori de –1.37%, eroare perfect acceptabilă. Dacă vom încerca să obținem aceeaşi viteză la 10MHz, cu brgh = high şi spbrg = 4 vom obține baud_rate_real = 125000bps ceea ce corespunde unei erori de +7.84%, rezultatul fiind inacceptabil. Schimbând cuarțul cu unul având 14.7456 MHz, pentru spbrg = 7 şi brgh = high vom obține baud_rate_real = 115278, ceea ce corespunde unei erori de +0.06 % şi este mai mult ca perfect. Cu speranța înțelegirii importanței calculului erorii în determinarea valorii registrului SPBRG, vom elucida mecanismul transmisiei via USART printr-un exemplu de comunicație [4] între PIC16F628 şi PC:

```
const xtal  = target_clock
const baudrate = 2400          -- modificați conform nevoilor proprii

procedure uart_init is        -- inițializarea modulului USART
-- ----------------------
  pin_b2_direction = output     -- tx
  pin_b1_direction = input      -- rx
  bank_1
assembler
  bcf txsta, tx9                -- modul cu 8 biți
  bsf txsta, txen               -- demarează transmisia
  bcf txsta, sync               -- selectează modul asincron
 -- bcf txsta, brgh             -- dezactivează high baud rate sau
  bsf txsta, brgh               -- activeză high baud rate
  bcf txsta, tx9d               -- curăță tx9d
end assembler

if brgh then  -- calculează spbrg, verificați rezultatul !
 spbrg = ( xtal / ( baudrate * 16 )) - 1
elsif ! brgh then
 spbrg = ( xtal / ( baudrate * 64 )) - 1
end if
; pragma test assert spbrg = xxx pragma test done
bank_0
assembler
  bsf rcsta, spen          -- activează modul serial
  bcf rcsta, rx9           -- pentru 8 biți
  bsf rcsta, cren          -- recepție constantă activată
```

```
  bcf rcsta, ferr          -- curăţă framing error
  movf rcreg, w            -- curăţă registrul de recepţie şi cele 2 fifo
  movf rcreg, w
  movf rcreg, w
  movlw 0
  movwf txreg              -- trimite orice "dummy" caracter
end assembler
end procedure

procedure async_rx ( byte out rx_data , bit out no_data_bit) is
  assembler
    local ser_in, uart_ready, no_int, overerror, frameerror, no_data
ser_in:
      btfsc  rcsta,oerr
      goto   overerror    -- tratare overflow error...
      btfsc  rcsta,ferr
      goto   frameerror   -- tratare framing error...
uart_ready:
      btfss  pir1,rcif
      goto   no_data      -- return bit pentru bufer gol
no_int:
      bcf    intcon_gie   -- dezactivează întreruperile
      btfsc  intcon_gie   -- asigură-te
      goto   no_int
      movf   rcreg,w      -- preia datele uart
      bsf    intcon_gie   -- activează întreruperile
      bsf    no_data_bit  -- s-a recepţionat ceva
      movwf  rx_data      -- salvează în registrul de recepţie
      return

overerror:                 -- rcreg este plin după al treilea bit recepţionat ?
      bcf    intcon_gie    -- dezactivează întreruperile
      btfsc  intcon_gie    -- asigură-te
      goto   overerror
      bcf    rcsta,cren    -- dezactivează recepţia continuă,resetare oerr
      movf   rcreg,w       -- curăţă rcreg + 2 fifo, ferr va fi resetat
      movf   rcreg,w
      movf   rcreg,w
      bsf    rcsta,cren    -- activează recepţia continuă, reset oerr
      bsf    intcon_gie    -- activează întreruperile
      goto   ser_in         -- reia de la capăt
frameerror:                -- dacă se recepţionează "gunoaie"
      bcf    intcon_gie    -- dezactivează întreruperile
      btfsc  intcon_gie
      goto   frameerror    -- fii sigur
      movf   rcreg,w      -- citirea rcreg curăţă ferr
      bsf    intcon_gie   -- activează întreruperile
      goto   ser_in       -- reia de la capăt
no_data:                   -- buferul FIFO este gol
      bcf    no_data_bit   -- ieşire din bucla de recepţie
      return
end assembler
end procedure
```

```
procedure async_tx ( byte in tx_data ) is
 assembler
  local transmit, interrupt
      movf   tx_data,w     -- copiază tx_data în w
transmit:
      btfss  pir1,txif
      goto   transmit      -- aşteaptă pentru flagul de întrerupere al  transmisiei
interrupt:
      bcf    intcon_gie    -- dezactivează întreruperile
      btfsc  intcon_gie    -- asigură-te
      goto   interrupt
      movwf  txreg         -- încarcă data de transmis
      bsf    intcon_gie    -- activează întreruperile
      return               -- data de transmis este în w
 end assembler
end procedure
```

In speranța că rutina prezentată îşi explică singură funcționarea prin comentariile prezente, precizările suplimentare ar fi:

- se poate renunța la dezactivarea/activarea întreruperilor ce intervin în rutinele de transmisie respectiv de recepție, dacă programul nu utilizează întreruperi,
- dacă se doreşte modificarea vitezei de comunicație atunci este necesară reinițializarea (prin lansarea procedurii uart_init) cu o valoare corespunzătoare în registrul SPBRG; reinițializarea trebuie făcută cu valoarea tuturor regiştrilor existentă după reset

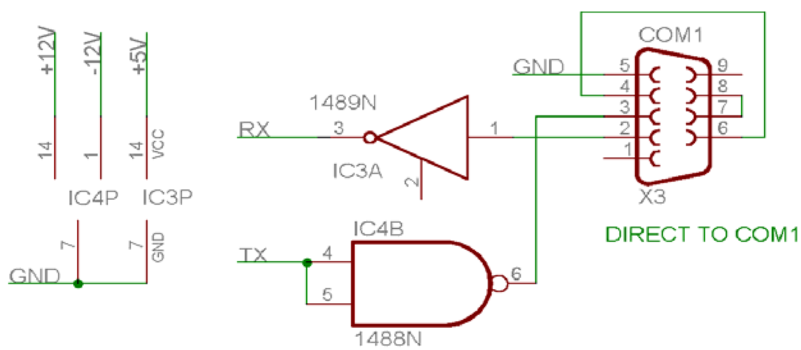


fig.6- 9 Convertoare de nivel clasice (1488/1489) pentru RS232

Cele mai ieftine şi accesibile convertoare de nivel datează din anii 80 (fig.6-9), sunt energofage şi necesită tensiuni de alimentare multiple. Dacă receptorul cvarduplu cu inhibare, 1489 are nevoie doar de +5V, emițatoarele 1488 se pot alimenta cu +5V şi –12V fie cu ±12V…±15V. La ora actuală au fost înlocuite cu circuitele integrate cu mecanism de generare proprie a tensiunii ridicate din linia de +5V utilizând structura *charge-pump,* prin care un oscilator local urmat de un multiplicator de tensiune încarcă/descarcă patru condensatoare electrolitice externe. Creşterea frecvenței acestui oscilator a făcut posibilă înlocuirea condensatorilor electrolitici cu gabarit mare cu condensatoare nepolarizate de 100nF, însă capabilitatea de curent a acestor drivere este redusă şi nici tensiunile generate în linie de transmițător nu depăşesc valoarea de +10V respectiv –10V în cazul cel mai bun. Acest tip de circuit integrat se pretează foarte bine la furtul de energie din liniile nefolosite ale interfeței seriale. Astfel se pot realiza uşor convertoare RS232 izolate galvanic față de sistemul care comunică şi care poate fi în contact fie cu tensiuni înalte, fie cu țesuturi umane supuse analizei sau stimulării (în aplicații medicale).

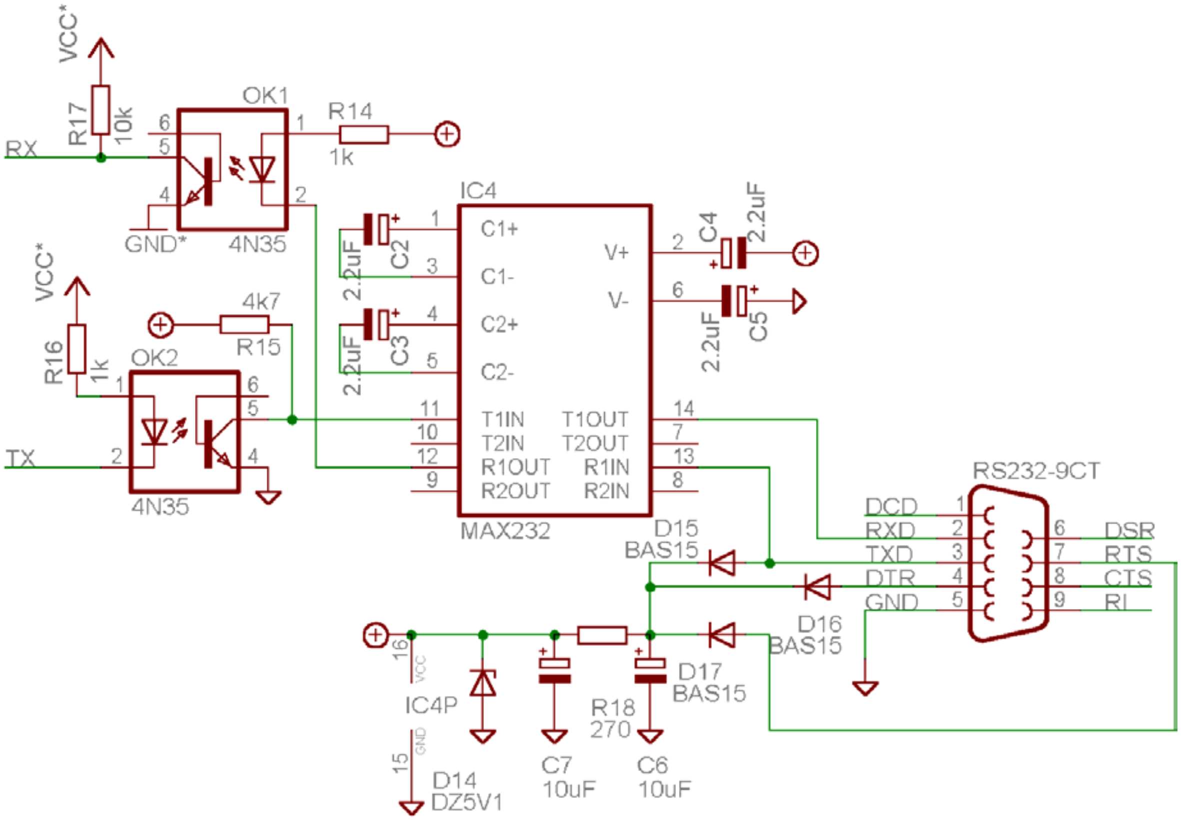


fig.6- 10 Interfață optoizolată RS232 alimentată prin "furt" de energie din PC

Interfața optoizolată din fig.6-10 utilizează liniile DTR şi RTS pentru obținerea celei mai importante părți din energia de alimentare şi TX pentru o cantitate infimă (când aceasta se găseşte în stare logică 0, standard RS232). Diodele sunt de comutație, tipul 1N4148 este perfect pentru acest scop. Se observă că optocuplorii asigură trecerea cu polaritate neinversată a semnalului TX respectiv RX spre PIC. Viteza de comunicație este limitată de MAX232 şi de viteza optocuplorilor. Interfața din figură a fost utilizată la maxim 4800 bps. Pentru obținerea vitezelor mai ridicate sunt necesari optocuplori de viteză, MAX232 având frecvența de transfer suficient de ridicată. Deşi datele de catalog pentru C2…C5 recomandă valoarea de 100nF sau1uF, utilizarea unor valori ceva mai mari (2.2uF) este posibilă şi benefică în acelaşi timp. Se recomandă utilizarea condensatorilor cu tantal. Pinii Rx şi TX ai USART la care se conectează semnalele RX şi TX din fig.6-10, diferă de la PIC la PIC:

PIC	**TX**	**RX**
16F628	8 (Rb2)	7 (Rb1)
16F877	25 (Rc6)	26 (Rc7)
16F876	17 (Rc6)	18 (Rc7)

6.2 Comunicația I^{2}C

Standardul I2C a fost descoperit acum 20…22 de ani de către Philips, în tentativa de a comunica între un microcontroler şi diverse circuite integrate cu specific audio şi video în aparatură electronică comercială (TV, vidorecorder, etc.) Denumirea este un acronim al cuvântului "**I**nter **IC** bus" printr-o prescurtare matematică a variantei anterioare: **IIC.** O primă informație ce derivă de aici este că lungimea liniei I2C nu este foarte lungă, fiind de regulă cel mult egală cu mărimea standardizată a cablajului imprimat sau în cazul cel mai defavorabil, când se utilizează caburi torsadate pentru transmisie între diverse module, de ordinul a 1…2 metri. Viteza de comunicație este 100KHz sau 400KHz (fast-mode) şi scade proporțional cu lungimea liniei şi modul de adaptare al terminatorului de linie (ce poate fi activ sau pasiv). Fiecare circuit conectat pe bus are adresă de identificare unică.

Transmisia se realizează cu trei fire SDA, SCL şi masă. **S**erial **DA**ta şi **S**erial **CL**ock sunt amândouă bidirecționale. Există două protocoale de funcționare: master-slave (stăpân-sclav) şi multimaster (mai mulți stăpâni). In modul master-slave circuitul integrat conectat pe bus care inițiază comunicația devine master (în cazul nostru PIC-ul), celelalte circuite răspunzând interogării ca slave. In modul multimaster două sau mai multe circuite integrate pot iniția comunicația pe rând, caz în care este nevoie de un algoritm de arbitrare pe bus.

In modul master-slave [5] inițierea comunicației se face cu comanda **start transmisie,** care este o secvență SDA = *low* urmată de SCL = *low*. In acest moment toate dispozitivele conectate pe bus aşteaptă o **adresă** (de 7 biți pentru standardul I2C sau de 10 biți pentru I2C fast-mode). Aceasta este transmisă împreună cu bitul de start într-o secvență de unul sau doi octeți. Dacă adresa unuia dintre sclavi se potriveşte, acesta va răspunde cu **ACK**nowledge, punând linia SCL = *low*, urmând să primească secvența de **date** formată din 8 biți, MSB fiind primul bit transmis. Dacă master-ul nu primeşte ACK, poate să blocheze transferul datelor trimițând o comandă de **stop transmisie.** De altfel terminarea transmisiei pe bus se face tot cu această comandă care înseamnă trecerea SCL = *high* urmată de SDA = *high.*

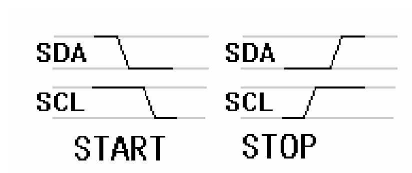


fig.6- 11 Comenzile START transmisie şi STOP transmisie pe bus-ul I2C

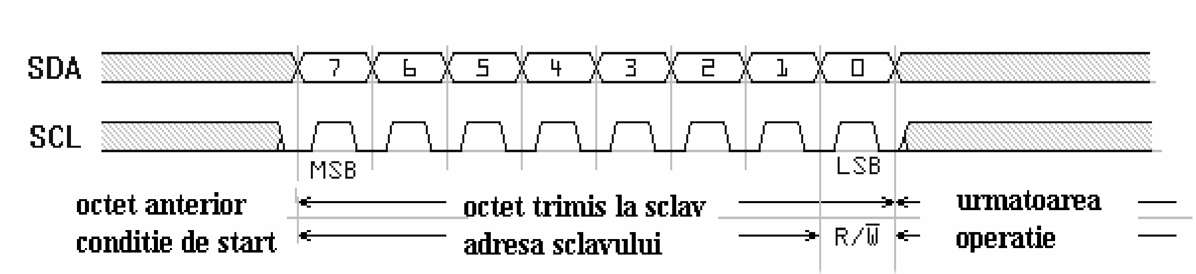


fig.6- 12 Transmisia unui octet spre sclav

Primul octet trimis după comanda START va identifica sclavul şi va selecta modul de operare [fig.6-12]. Conținutul celorlalți octeți va fi dependent de răspunsul sclavului. Dacă pe bus sunt sclavi care au 10 biți de adresă, ei vor răspunde toți la semnalul ACK inițiat de master. Următorul octet transmis de master va fi luat în considerare şi valoarea acestuia evaluată pentru determinarea adresei interogate. Şi în acest mod extins cu adresă de 10 biți, primul bit după comanda START va determina modul de acces al sclavului (1 = citeşte, 0 = scrie). Odată ce sclavul a fost adresat şi acesta a răspuns cu ACK, poate fi recepționat un octet de la sclav dacă bitul R/W din adresă a fost setat 1. Protocolul de citire este identic cu cel de transmisie a unui octet spre sclav, cu precizarea că master-ul nu mai controlează linia SDA, ci generează un front crescător pe SCL (2)) [fig.6-13], citeşte nivelul logic pe SDA (3) şi generează un front descrescător pe SCL (4). Sclavul nu va schimba datele atât timp cât SCL = *high*, altfel pot fi generate condiții false urmate de generarea comenzii START/STOP.

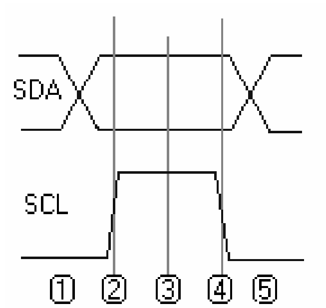


fig.6- 13 Algoritmul de citire a datelor dinspre sclav cu generarea concomitentă a tactului

Algoritmul prezentat în fig.6-13 este repetat de 8 ori pentru a se obține un octet de date [fig.6-14]. Semnificația înformației acestui octet depinde numai de sclav, iar bitul care se transmite primul este întotdeauna MSB. De aceea pentru interpretarea corectă a rezultatului trebuie citită cu atenție fila de catalog a sclavului specific care se utilizează.

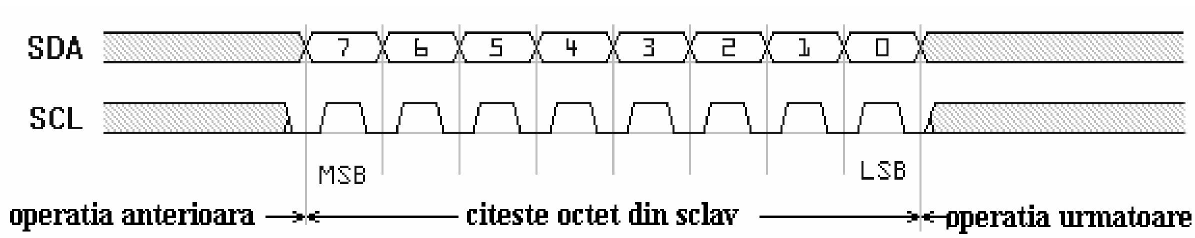


fig.6- 14 Generarea cuvântului de către sclav

Am observat că răspunsul sclavului după recepționarea unui octet de adresă sau de date se face prin ACKnowledge. Acest lucru înseamnă punerea liniei SDA în stare *low* imediat după recepția celor 8 biți transmişi, sau în cazul recepției adresei, imediat după evaluarea valorii adresei.

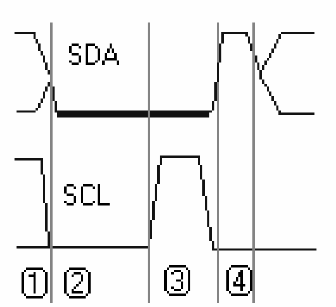


fig.6- 15 Generarea acknowledge-lui de către sclav

Imediat ce master-ul pune SCL în stare *low* pentru a termina transmisia bitului (1)[fig.6-15], SDA va fi pusă în stare *low* de către sclav (2), master-ul va genera un tact pe SCL (3) iar sclavul va elibera linia SDA înainte de terminarea tactului (4). Bus-ul este din nou disponibil pentru master, ca acesta să continue să transmită date sau să genereze o comandă STOP. In cazul unei date scrise în sclav, ciclul trebuie completat

înaintea generării unei condiții de stop. Sclavul va bloca bus-ul ținând linia SDA în stare *low*, până când masterul a generat un impuls de tact pe SCL. In mod analog, după recepția unui octet transmis de sclav, master-ul trebuie să aducă la cunoştința sclavului, prin acknowledge, acest lucru. In acest moment, master-ul deține controlul total asupra liniilor SDA şi SCL.

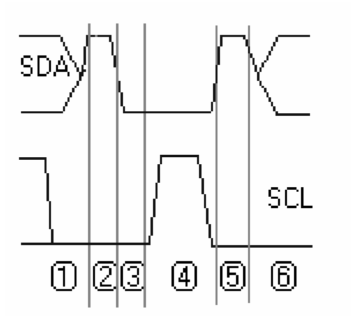


fig.6- 16 Generarea acknowledge-lui de către master

După transmisia ultimului bit spre master (1) [fig.6-16], sclavul va elibera controlul asupra liniei SDA, care va trece în stare logică *high* (2). Master-ul va trage acum linia SDA în stare *low* şi va genera un tact pe SCL (4). După terminarea acestui impuls de tact, master-ul va elibera din nou linia SDA (5) în timp ce sclavul va prelua din nou controlul asupra ei (6). In mod real stările (2) şi (5) explicate aici nu sunt vizibile pe un osciloscop fără memorie, având durate foarte scurte. Răspunsul cu acknowledge după orice octet primit de la sclav este obligatoriu, cu excepția ultimului octet. Dacă master-ul doreşte să oprească recepția datelor de la sclav, trebuie să poată trimite o comandă de stop. Deoarece sclavul preia controlul liniei SDA după acknowledge-ul generat de master, în acest moment pot apare probleme: să presupunem că următorul bit pregătit pentru transmisie spre master este 0. Linia SDA va fi trasă în stare *low* de către sclav, imediat după ce masterul trage linia SCL *low*. Master-ul este pregătit acum să trimită o comandă de stop pe bus. Eliberează întâi linia SCL şi apoi încearcă să elibereze linia SDA care este ținută în stare *low* de către sclav. Concluzia este că nu a putut fi generată comanda de stop pe bus. Această situație se numeşte **N**ot**ACK**nowledge şi nu trebuie confundată cu **NO ACK**nowledge. In timp ce NACK poate apare după ce master-ul a citit un octet de la sclav, NOACK poate apare după ce masterul a scris un octet spre sclav. NOACK este o stare ce poate apare pe parcursul curgerii datelor dintre stăpân şi sclav. Dacă după transmisia celui de-al 8-lea bit dinspre master spre sclav, sclavul nu pune SDA *low*, aceasta este considerată o condiție NOACK. Acest lucru poate însemna implicit:

- Sclavul nu este prezent (sau nu are adresa validă)
- Sclavul pierde un puls şi iese din sincronizarea generată de master pe SCL
- Bus-ul este defect, una din linii fiind în permanență scurtcircuitată la masă

Bus-ul I2C se bazează pe o structură de ieşire a circuitului integrat cu tranzistor open-drenă sau open-colector, respectiv de intrare prin buffer. Când linia este inactivă, ea se găseşte în stare logică *high*. Tranzistorul cu drena în vânt necesită o rezistență de sarcină care se găseşte amplasată în capătul liniei spre +5V. Pentru a transfera informație pe bus, circuitul respectiv trage linia în stare logică *low*. Dacă lungimea liniei este mare, aceasta va avea o capacitate parazită importantă. Adăugând şi capacitățile parazite interne ale fiecărui integrat conectat pe bus, cu cât numărul acestora este mai mare, constanta de timp a liniei (RC) este mai mare şi implicit frecvența de transfer a datelor mai scăzută. Efectul vizibil pe un osciloscop conectat pe bus este "înmuierea" fronturilor de comutare. Un terminator rezistiv poate fi folosit pentru capacități ale liniei sub 200pF. Dacă adaptarea terminatorului la linie nu este corectă pot apare oscilații care se "plimbă" pe bus.

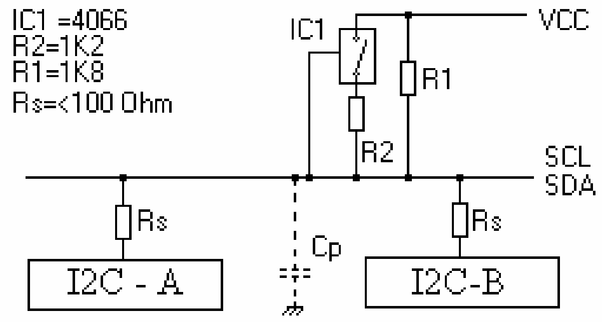


fig.6- 17 Implementarea unui terminator activ pentru bus I2C

Efectul vizibil pe osciloscop este oscilația fronturilor semnalelor sau jiter. Pentru linii cu capacitate mai mare de 200pF, funcționând la 400kHz e nevoie de terminatoare active. Acestea sunt construite în jurul a două sau trei tranzistoare cu efect de câmp a căror rol este de a scădea rezistența de pull-up în situația prezenței unei capacități parazite (distribuite pe bus) de valoare mare. In momentul când condensatorul parazit s-a încărcat, valoarea rezistenței este crescută din nou pentru a minimiza consumul suplimentar de curent pe bus. Un astfel de terminator activ se poate realiza cu ajutorul unui comutator CMOS MMC4016, MMC4066 sau al orcărui multiplexor MMC405x. [fig.6-17] Rezistențele Rs (47…82ohm), în serie cu circuitele integrate conectate la bus protejează intrările acestora la supracreşteri de tensiune. Presupunând că linia este inactivă (SDA, SCL sunt în stare logică 1) şi că IC1 nu este montat în circuit, timpul de încărcare al capacității parazite a liniei Cp depinde doar de valoarea rezistenței R1. Cu cât aceasta este mai mare, cu atât va fi nevoie de o perioadă mai lungă de timp pentru încărcarea Cp. La 200pF şi 1K8, timpul de încărcare va fi de aproximativ 360nS, mai mare decât limita prevăzută de 300nS pentru I2C rapid, curentul de încărcare fiind de cca 3mA. In momentul în care tensiunea pe bus se găseşte în zona 0.8-2V, comutatorul se închide şi pentru o scurtă perioadă de timp, creşte curentul la 5V/R1||R2 adică la 7mA, rezultând o încărcare completă a Cp în mai puțin de 300nS. Când tensiunea pe bus s-a stabilizat, comutatorul se deschide scăzând curentul la valoarea inițială.

In modul multimaster două sau mai multe circuite integrate pot iniția comunicația pe rând, caz în care este nevoie de un algoritm de arbitrare pe bus. Acest mod de funcționare este ceva mai complicat, pentru a evita ca datele transferate să fie interpretate greşit. Presupunem că pe bus se găsesc doi masteri şi mai mulți sclavi. Dacă master-ul A generează o comandă de START, şi trimite o adresă pe bus, toți sclavii (inclusiv master-ul B care poate fi considerat în acest moment sclav) vor aştepta. Dacă adresa nu se potriveşte cu cea a master-ului B, acesta trebuie să înceteze orice activitate pe bus până când acesta devine din nou liber ca urmare a unei comenzi STOP. Cât timp cel de-al doilea master nu face altceva decât să monitorizeze linia, totul este OK. Problemele încep când unul dintre masteri pierde secvența de START şi crede că bus-ul e liber . De aceea orice master care schimbă starea liniei în *high,* trebuie să verifice că într-adevăr comanda respectivă a schimbat starea liniei în *high*. Dacă linia rămâne totuşi *low*, înseamnă că alt master are controlul asupra liniei. Regula generală va fi: dacă un master nu poate trece linia în stare *high,* înseamnă că a pierdut controlul asupra bus-ul şi va rămâne inactiv până la următoarea comandă STOP, fără a încerca să trimită altă comandă de START. Această regulă va împiedeca orice interpretare eronată a datelor, deoarece nici un master nu poate conturba

activitatea altui master şi neputând detecta trecerea uneia dintre liniile bus-ului în stare *high*, va trece imediat în stare inactivă.

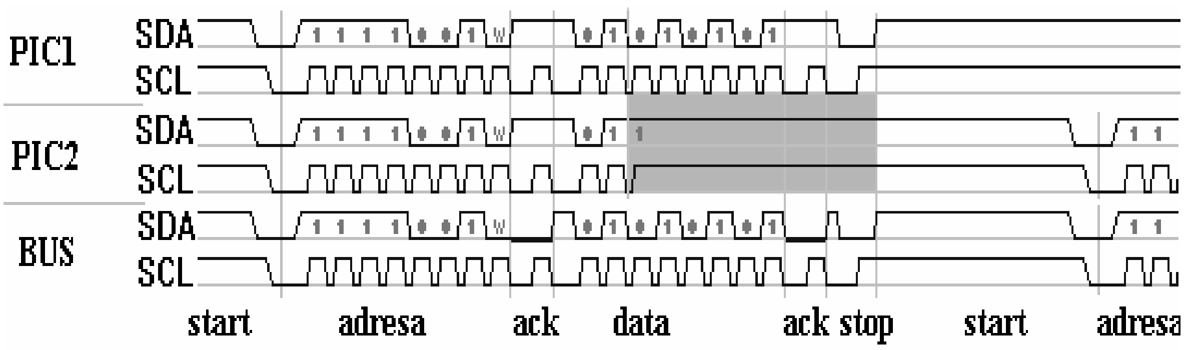


fig.6- 18 Arbitrarea bus-ului în modul multimaster

Exemplul de arbitrare a bus-ului este prezentat în fig.6-18. In acest exemplu s-a considerat că două microcontrolerele (notate ca PIC1 şi PIC2), scriu pe bus la adresa 1111001. La această tentativă, slave-ul răspunde cu ACK. Până în acest moment amîndouă PIC-urile au impresia că dețin controlul pe bus. PIC1 doreşte să trimită 01010101 spre sclav în timp ce PIC2 doreşte să transmită 01100110. In momentul în care datele nu mai sunt identice, deoarece PIC-ul trimite ceva diferit de ceea ce este pe bus, unul dintre PIC-uri va pierde controlul bus-ului şi va deveni inactiv (acesta va fi cel care nu-şi va regăsi datele transmise pe bus). Atât timp cât nu s-a generat nici o comandă STOP pe bus, liniile SDA şi SCL ale acestuia rămân flotante (zona haşurată din fig.6-18). In momentul în care a fost detectat un STOP, PIC2 poate încerca să transmită din nou. Din exemplul de mai sus putem trage concluzia că PIC-ul care ține bus-ul low într-o situație arbitrară, este cel care câştigă întotdeauna controlul bus-ului, în timp ce PIC-ul care doreşte ca bus-ul să fie *high* în timp ce acesta este ținut *low* de celălalt PIC, pierde controlul asupra bus-ului. Când un PIC pierde controlul asupra bus-ului are nevoie obligatorie de apariția unei comenzi STOP pe bus; în acest moment el ştie că transmisia anterioară a luat sfârşit.

Ultima mare realizare în standardul I2C este evoluția acestuia spre standardul ultra-rapid, care permite transferul datelor la cca. 3 Mbps, putând menține şi compatibilitatea cu standardul rapid sau normal. In standardul ultra-rapid, biții de adresă se transmit organizați pe doi octeți, primul conține rezerva de adresă a standardului extins şi primii doi biți semnificativi din adresa curentă, iar al doilea octet conține biții mai puțin semnificativi ai adresei curente.

In capitolul 2 cititorul a văzut cum se pune problema transferului datelor prin I2C, studiind exemplul de interfațare al circuitului LM75. Mecanismul de funcționare al magistralei I2C fiind dezvăluit în detaliu, exemplul tipic de interfațare la microcontroler este eepromul I2C. Deşi PIC-urile flash dispun de eeprom intern, este foarte posibil fie ca dimensiunea memoriei acestuia să nu ajungă, fie ca utilizatorul să scrie în mod repetat, cu o frecvență ridicată la aceleaşi adrese de memorie şi având sentimentul că acestea se vor distruge în timp, să intenționeze stocarea datelor într-un eeprom extern. Aşa cum v-am obişnuit şi în cazul comunicației SPI, există două moduri de scriere în eeprom: prin algoritm software (pentru toate PIC-urile) şi prin algoritm hardware (numai PIC-urile ce dispun de interfață **M**aster **S**ynchronous **S**erial **P**ort).

6.2.1 Adresarea memoriei eeprom seriale cu interfață I2C

Memoria eeprom [6] (CD:/datasheet/microchip/24C04.pdf) este alcătuită dintr-o matrice de tranzistoare MOS a căror capacitate poate fi încărcată sau ştearsă electric şi o logică de comandă ce acționează direct asupra un buffer **F**irst **I**n **F**irst **O**ut, a unui registru *pointer* (index) şi care generează intern tensiunea de programare Vpp (fig.6-19).

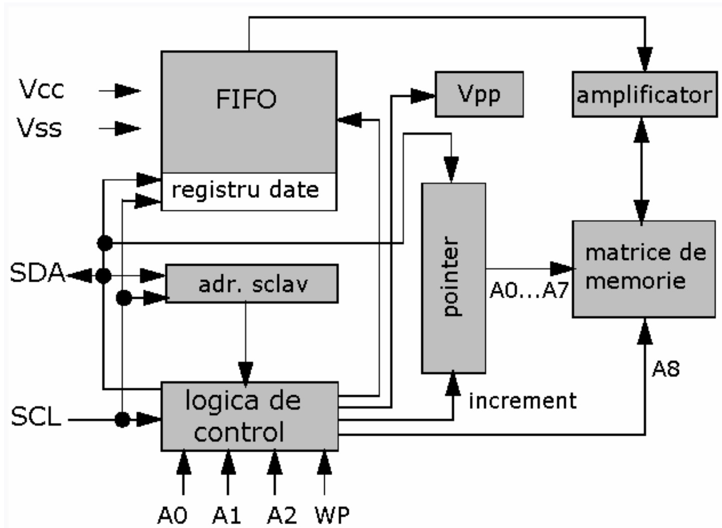


fig.6- 19 Memorie eeprom-schemă bloc

Memoria comunică în exterior cu SDA şi SCL având semnificația descrisă în subcapitolul anterior, trei linii de adresă pentru configurare hardware şi un pin de protecție la scriere (WP=0 permite citirea/scrierea întregii memorii, WP=0 bancul superior de memorie 256KB este protejat la scriere). Memoria eeprom nu poate fi decât sclav pe bus-ul I2C. Adresa acesteia are un format standardizat pentru toate tipurile de eepromuri I2C (fig.6-20).

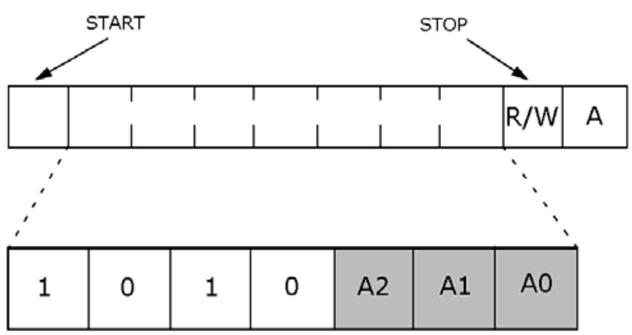


fig.6- 20 Semnificația biților de adresă a memoriei sclav

Primii patru biți (1010) reprezintă familia I2C de memorii eeprom, A2, A1 şi A0 sunt pini de validare a selecției memoriei pentru o adresă dată. Memoria nu va fi accesată dacă biții respectivi din cuvântul de adresă nu respectă valoarea logică a semnalelor electrice existente pe liniile A2, A1 şi A0. De exemplu, dacă memoria este conectată astfel: A2=0, A1=1, A0=0, cuvântul corect de adresă va fi: 1010_0100 pentru scriere în eeprom (R/W = 0) respectiv 1010_0101 pentru citire din eeprom (R/W=1). Un eprom cu trei linii de selecție este 24C256. Rezultă că se pot conecta pe un bus I2C, maxim 8 astfel de memorii (2^3 = 8

unde 3 = nr. de stări posibile, 2 = baza sistemului de numerație). Unele memorii organizate pe două blocuri de memorie, au doar două astfel de linii de adresă, A0 având rolul de selecție internă a blocului de memorie (24C04, A0=0 blocul 0-0FF, A0=1 blocul 100-1FF). Pentru acestea A0=0. Din punct de vedere al organizării matricii interne, există memorii de 8 biți sau de 16 biți. Algoritmul de citire/scriere a unei locații de memorie este foarte asemănător pentru ambele tipuri, la cele de 16 biți repetându-se secvența pentru fiecare din cei doi octeți de adresă :

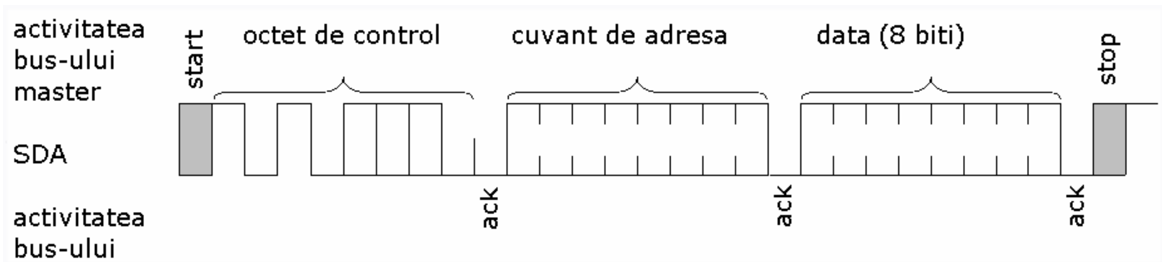


fig.6- 21 Exemplu de scriere (R/W = 0) a unui octet în eeprom de 8 biți

Modul de generare a semnalelor de start, stop, ack, nack este prezentat în biblioteca i2cm.jal. Secvența de scriere începe întotdeauna cu un start, transmisia octetului de control corelat cu tipul de operație ce urmează (citire sau scriere), adresa şi data. După fiecare octet recepționat memoria răspunde cu ack, dacă se doreşte încheierea secvenței este necesară şi comanda de stop. Citirea unei adrese curente are un algoritm asemănător:

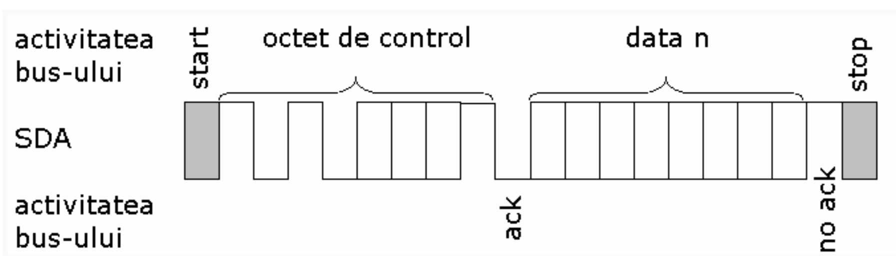


fig.6- 22 Exemplu de citire a adresei curente în eeprom de 8 biți

Această operație este necesară deoarece după fiecare scriere în eeprom, indexul de adresă se incrementează automat necesitând repoziționarea lui la citire. Dacă memoria are 2 octeți de adresă, se vor succeda cele două valori ale MSB şi LSB (dublarea secvenței DATA n din fig.6-22) cu acknowledge-ul corespunzător după MSB. Terminarea secvenței se face obligatoriu cu nack şi stop. Este evident că e nevoie întâi de o scriere, DATA n (fig.6-22) fiind adresa de la care se citeşte, urmată de aceeaşi secvență în care octetul de control este setat pentru citire, DATA n fiind acum data memorată la adresa accesată anterior. Memoriile eeprom dispun şi de o metodă mai rapidă de scriere/citire la nivel de pagină (8 sau 16 biți) algoritmul asemănător cu cel prezentat anterior, se găseşte în orice filă de catalog pentru memoria eeprom utilizată. Pe capsula memoriei este marcată explicit capacitatea memoriei măsurată în biți şi nu în octeți: 24C04 = 4096biți = 4Kbit/8 = 512 octeți.

In fig.6-23 sunt interfațate două memorii 24C04-PIC, utilizându-se toate liniile de comandă ale acestora, mai puțin protecția la scriere (WP=0, citirea şi scrierea sunt posibile, WP=1 scrierea în bancul superior de memorie este interzisă). Din punct de vedere logic este aberant să utilizăm două astfel de memorii în configurația prezentată, deoarece există memorii cu capacitate dublă (24C08). Din punct de vedere educațional, exemplul arată cum se pot interfața orice două circuite I2C cu rol de sclav pe acelaşi bus.

6.2.2 Interfațarea eeprom-ului I2C la PIC prin algoritm software

Indiferent de modul de abordare al interfațării, este importantă cunoaşterea precisă a modului de funcționare al eepromului ales. Copierea rutinelor dedicate unor memorii pentru a fi utilizate cu alte tipuri de memorii, dau de obicei rezultate negative, atât timp cât memoriile au alt algoritm de scriere/citire. Deci nu încercați din start acestă metodă "oarbă" de interfațare chiar dacă eepromul are aceeaşi denumire dar producători diferiți !

Ambele linii ale bus-ului I2C sunt bidirecționale şi necesită terminatoare rezistive spre Vcc, datorită configurației specifice de intrare-ieşire utilizată de biblioteca i2cm, care este definită în i2cp.jal:

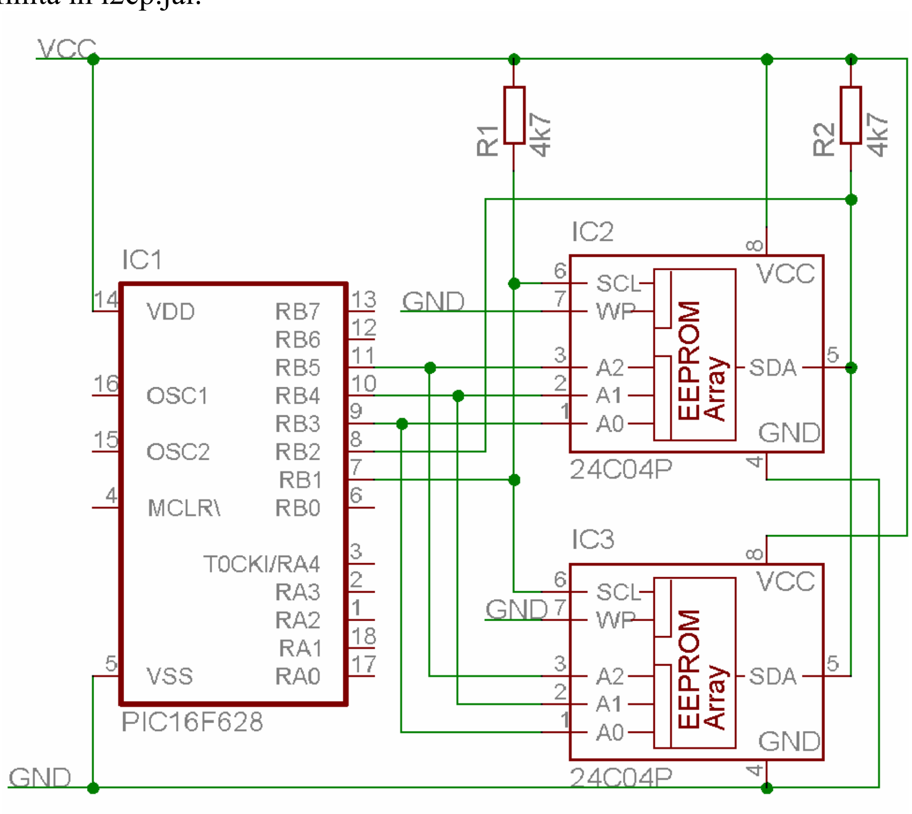


fig.6- 23 Interfațarea memoriilor 24C04 la PIC cu utilizarea totală a liniilor de adresă

```
var volatile bit i2c_clock_in   is pin_b1        ; b1 = CLK
var volatile bit i2c_clock_out  is pin_b1_direction
var volatile bit i2c_data_in    is pin_b2        ; b2 = SDA
var volatile bit i2c_data_out   is pin_b2_direction
```

Se observă modul de manipulare al liniilor de tact şi date în mod *open drenă* (chiar dacă liniile respective sunt ieşiri MOS standard), astfel încât, pentru situația în care liniile sunt intrări, nivelul logic *high* este asigurat doar de rezistențele de *pull-up*. Programul ce efectuează comunicația este următorul, biblioteca i2cm (i2c**m**odificată) se găseşte în distribuția JAL de pe CD.

```
include 16F628_20
include jpic
include i2cm

; eeprom 24C04
const byte eeprom1_adr = 0b_1010_0000 ; A2,A1 = 0, A0 = 0, bloc0
const byte eeprom2_adr = 0b_1010_1110 ; A2,A1 = 1, A0 = 1, bloc1
; adresele celor 2 sclavi

var byte test_data1, test_data2
var volatile bit A2 is pin_b5
pin_b5_direction = output
var volatile bit A1 is pin_b4
pin_b4_direction = output
var volatile bit A0 is pin_b3
pin_b3_direction = output

; _i2c_init              ; inițializarea comunicației I2C
; i2c_put_stop           ; este realizată în i2cm.jal
; _i2c_wait

 ; configurează hardware adresarea memoriei eeprom1
 A2 = low A1 = low A0 = low
 ; scrie la locația 0x00 a memoriei eeprom1, valoarea 0xAA
 i2c_write_2( eeprom1_adr , 0x00 , 0xAA )
 ; adresează locația ce va fi citită ( adresa curentă )
 i2c_write_1( eeprom1_adr , 0x00 )
 ; citeşte locația 0x00 , valoarea citită este 0xAA ?
 i2c_read_1 (  eeprom1_adr , test_data1 )

 ; configurează hardware adresarea memoriei eeprom2
 A2 = high A1 = high A0 = high
 ; scrie în ultima locație a memoriei eeprom1, valoarea 0xEE
 i2c_write_2( eeprom2_adr , 0xFF , 0xEE )
 ; adresează locația ce va fi citită ( adresa curentă )
 i2c_write_1( eeprom2_adr , 0xFF )
 ; citeşte locația 0x00 , valoarea citită este 0xEE ?
 i2c_read_1 (  eeprom2_adr , test_data2 )

 if test_data1 == 0xAA then
   ; operație de semnalizare "succes" la discreția utilizatorului
 elsif test_data2 == 0x_EE then
   ; idem, semnalizare succes
 else  ; semnalizare eşec
 end if

 end
```

De notat că liniile de adresă ale eepromurilor nu este necesar să fie conectate la PIC, ele pot fi conectate direct la GND şi VCC aşa cum logica o cere.

6.2.3 Interfațarea eeprom-ului I2C la PIC prin algoritm hardware

Modulul intern al PIC-ului responsabil pentru transfer I2C hardware este MSSP. **M**aster **S**ynchronous **S**erial **P**ort este un modul care pare atât la prima cât şi la o a doua privire, complicat, încurcat, destinat unui utilizator cu un dezvoltat simț al răbdării. Vom analiza doar funcționarea acestuia în modul I2C master. Sunt nici mai mult nici mai puțin de şase regiştrii utilizați în modul I2C, doi regiştrii de control SSPCON şi SSPCON2, un registru status SSPSTAT, registrul de stocare a datei de transmis sau de recepționat SSPBUF, un registru de serializare SSPSR care din fericire nu este direct accesibil şi un registru de adresă SSPADD. Pentru a simplifica puțin lucrurile, regiştrii vor fi descrişi în conformitate cu rolul lor ocupat în rutinele ce realizează comunicația cu o memorie eeprom 24LC21 [7] (1024 biți = 1Kbit). Trei pini sunt utilizați pentru a interfața această memorie: **W**rite **E**nable, **S**erial **CL**ock şi **S**erial **Da**ta. W_EN în 1 logic permite scrierea în memoria eeprom, ceilalți doi pini au funcțiile descrise anterior.

```
pin_c3_direction = input  -- SCL este pin_c3 al PIC-ului
pin_c4_direction = input  -- SDA este pin_c4 al PIC-ului
pin_c5_direction = output -- W_EN este pin_c5 al PIC-ului
```

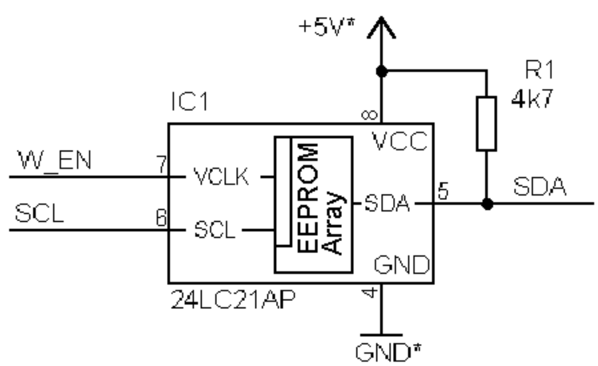


fig.6- 24 Configurația pinilor eepromului

```
procedure i2c_init is -- inițializarea comunicației
    bank_1
   f877_sspstat = 0b_1000_0000 -- slew rate control dezactivat
   f877_sspcon2 = 0b_0110_0000 -- setarea acknowledge-ului
-- baud rate = fosc/(4x(f877_sspadd + 1 ))
   if target_clock == 4_000_000 then -- viteza de comunicație 100kHz
      f877_sspadd  = 9
    elsif target_clock == 10_000_000 then
      f877_sspadd  = 24
    elsif target_clock == 20_000_000 then
      f877_sspadd  = 49
    else pragma error ; oscilator cu altă frecvență !
   end if
   bank_0
   pin_c5 = high
```

```
-- activează scrierea în eeprom ( pin_c5=low, permite doar citirea )
   f877_sspcon   = 0b_0010_1000 ; portul serial in mod i2c master
end procedure
```

Inițializarea modulului se face scriind viteza de comunicație standard (100KHz/1MHz, SMP=1) sau cea rapidă (400KHz, SMP=0) [fig6-25], în registrul SSPSTAT. Pentru o mai bună inteligibilitate, toți regiştrii prezentați vor fi explicați doar pentru modul I2C al MSSP. Registru SSPCON2 [fig.6-26], este destinat modului de setare al răspunsului prin ACKDT respectiv pentru inițierea secvenței acknowledge pe pinii SDA şi SCL prin ACKEN, generarea comenzii de start prin bitul SEN şi cea de stop prin PEN, generarea comenzii de restartare prin RSEN şi activarea recepției prin RCEN. In tabelul de mai jos sunt prezentate comparativ rutinele de start, restart şi stop scrise în limbaj assembler sub topică JAL (stânga), respectiv aceleaşi rutine în limbaj JAL pur (dreapta).

Rutine în assembler sub JAL	**Rutine în JAL pur**
procedure i2c_start_asm is bank_1 assembler local l1 BSF SEN l1: BTFSC SEN GOTO l1 end assembler bank_0 end procedure	procedure i2c_start_jal is bank_1 SEN = high while SEN loop end loop bank_0 end procedure
procedure i2c_restart_asm is bank_1 assembler local l2 BSF RSEN l2: BTFSC RSEN GOTO l2 end assembler bank_0 end procedure	procedure i2c_restart_jal is bank_1 RSEN = high while RSEN loop end loop bank_0 end procedure
procedure i2c_stop_asm is bank_1 assembler local l3 BSF PEN l3: BTFSC PEN GOTO l3 end assembler bank_0 end procedure	procedure i2c_stop_jal is bank_1 PEN = high while PEN loop end loop bank_0 end procedure

SMP	CKE	D/A	P	S	R/W	UA	BF
7 R/W	6 R/W	5 R	4 R	3R	2 R	1 R	0 R
SMP: bitul de eşantionare Mod I2C stăpân sau sclav: 1 = control de viteză dezactivat pentru viteza standard (100KHz şi 1MHz) 0 = control de viteză activat pentru mare viteză (400KHz)							
CKE: selecția frontului tactului SPI Mod I2C stăpân sau sclav: 1 = nivel de intrare conform specificației SMBUS 0 = nivel de intrare conform specificației I2C							
D/A: data sau adresa 1 = indică faptul că ultimul octet recepționat a fost data 0 = indică faptul că ultimul octet recepționat a fost adresa							
P: bitul de stop (bit resetat când MSSP este dezactivat, SSPEN resetat) 1 = bitul de stop a fost ultimul detectat (acest bit este 0 la reset) 0 = bitul de stop nu a fost detectat							
S: bitul de start (bit resetat când MSSP este dezactivat, SSPEN resetat) 1 = bitul de start a fost ultimul detectat 0 = bitul de start nu a fost detectat							
R/W: bitul de informație pentru operația de citire/scriere (acest bit este valid de la ultima adresare până la următoarea operație de start, stop sau not ackn) Mod I2C sclav: 1 = citeşte 0 = scrie Mod I2C stăpân: 1 = transmisie în desfăşurare 0 = transmisia nu este în desfăşurare							
UA: adresă extinsă (numai în mod I2C cu 10 biți de adresă) 1 = utilizatorul trebuie să modifice adresa în registrul SSPADD 0 = adresa nu trebuie modificată							
BF: bitul de status al bufferului La recepție, mod I2C: 1 = recepția este completă, SSPBUF este plin 0 = recepția nu e completă, SSPBUF este gol La transmisie, mod I2C: 1 = transmisia de date este în desfăşurare (fără biții de stop şi not ack), SSPBUF este plin 0 = transmisia s-a terminat (fără biții de stop şi not ack), SSPBUF este gol SSPSTAT-i2c							

fig.6- 25 Configurația registrului SSPSTAT pentru modul I2C

GCEN	ACKSTAT	ACKDT	ACKEN	RCEN	PEN	RSEN	SEN
7 R/W	6 R/W	5 R/W	4 R/W	3R/W	2 R/W	1 R/W	0 R/W

GCEN: bitul pentru setarea adresării universale (numai modul I2C sclav)
1 = setează întreruperea la o recepţie în SSPSR a unei comenzi 0000h
0 = adresarea universală este dezactivată

ACKSTAT: bitul status al ack (numai în mod I2C stăpân, transmisie)
1 = nu a fost recepţionat ack de la sclav
0 = a fost recepţionat ack de la sclav

ACKDT: bitul de semnalizare al ack (numai în mod I2C stăpân)
Reprezintă valoare ce va fi transmisă când utilizatorul initiază o secvenţă ack la sfârşitul recepţiei 1 = nack, 0 = ack

ACKEN: bitul de validare al secvenţei de ack (în mod I2C stăpân, recepţie)
1 = iniţiază secvenţa ack pe pinii SDA, SCL şi transmite ACKDT, resetat hardware
0 = fără secvenţă ack

RCEN: bit de setare a recepţiei (numai mod I2C stăpân)
1 = activează modul de recepţie I2C
0 = recepţia este inactivă

PEN: bit de setare a condiţiei de stop (numai I2C stăpân)
1 = iniţiază condiţia de stop pe SDA şi SCL, resetat hardware
0 = condiţia de stop inactivă

RSEN: bit de setare a condiţiei de restart (numai mod I2C stăpân)
1 = iniţiază condiţia de restart pe SDA şi SCL, resetat hardware
0 = condiţia de restart inactivă

SEN: bit de setare a condiţiei de start (numai mod I2C stăpân)
1 = iniţiază condiţia de start pe SDA şi SCL, resetat hardware
0 = condiţia de start inactivă SSPCON2

fig.6- 26 Registrul SSPCON2

Revenind la rutina de iniţializare, bitul SSPEN din registrul SSPCON (fig.6-27), este cel care iniţializează portul serial şi configurează SDA şi SCL ca pini ai portului serial, în timp ce SSPM3…SSPM0 setează modul de lucru al interfeţei (în cazul nostru master, cu frecvenţa de tact = Fosc/(4*(SSPADD+1)). Rutinele de start, restart şi stop, testează biţii corespunzători din registrul SSPCON2 (fig.6-26). Cele două modalităţi de abordare a rutinelor sunt un exerciţiu comparativ între limbajul de asamblare şi limbajul pur JAL, funcţionalitatea finală fiind identică (nu acelaşi lucru putem spune despre dimensiunea codului hexa generat în urma compilării, care este ceva mai mare pentru rutinele în limbaj Jal pur).

```
procedure i2c_transmit ( byte in data ) is
   f877_sspbuf = data
   bank_1
    assembler
    local l4
  l4:   BTFSC R_W
```

```
       GOTO   l4
       BCF    status_C
       BTFSC  ACKSTAT
       BSF    status_C
    end assembler
  bank_0
end procedure

procedure i2c_receive ( byte out data ) is
  bank_1
  ACKDT = low
  assembler
   local l5, l6
       BTFSC  status_C
       BSF    ACKDT
       BSF    RCEN
 l5:   BTFSS  BF
       GOTO   l5
       BSF    ACKEN
 l6:   BTFSC  ACKEN
       GOTO   l6
  end assembler
  bank_0
  data = f877_sspbuf
end procedure

procedure write_24C21 ( byte in address, byte in data ) is
  i2c_start_asm
  i2c_transmit ( 0xa0 )  -- octet de comandă, scrie
 if status_c then i2c_stop end if -- NoACK eroare
  i2c_transmit ( address )
  i2c_transmit ( data )
  i2c_stop_asm
end procedure

procedure read_24C21 ( byte in address , byte out data ) is
  i2c_start_asm
  i2c_transmit ( 0xa0 )  -- octet de comandă, scrie
  i2c_transmit ( address )
  i2c_restart_asm
  i2c_transmit ( 0xa1 )  -- octet de comandă, citeşte
  i2c_receive ( data )
  i2c_stop_asm
end procedure
```

O inspecție detaliată a rutinelor de mai sus, evidențiază faptul că acestea nu pot fi utilizate în întreruperi deoarece nu este utilizat bitul SSPIE (registrul PIE1), pentru activarea întreruperii I2C, respectiv bitul SSPIF (registrul PIR1), pentru citirea evenimentului (ambii regiştrii sunt prezentați în capitolul 5). De asemenea ele nu pot fi utilizate la accesarea memoriilor eeprom cu adresa organizată pe doi octeți. De aceea, o altă variantă funcțională a programului de citire/scriere a eepromului este prezentată în continuare:

WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
7 R/W	6 R/W	5 R/W	4 R/W	3R/W	2 R/W	1 R/W	0 R/W

WCOL: bitul de detcție al coliziunii la scriere
Mod stăpân: 1 = s-a încercat scrierea în SSPBUF cât timp condițiile I2C nu erau valide
0 = nu a fost coliziune
Mod sclav: 1 = SSPBUF este scris cât timp se transmite cuvântul anterior (ştergere soft)
0 = nu a fost coliziune

SSPOV: bitul de semnalizare a depăşirii la recepție
1 = un octet este receptionat cât timp SSPBUF menține octetul anterior (ştergere soft)
0 = nu este depăşire

SSPEN: bitul de setare al portului sincron serial
1 = activează portul serial şi configurează SDA, SCL ca pini ai portului serial
0 = dezactivează portul serial, SDA,SCL devin pini IO

CKP: bitul de selecție al polarității semnalului (folosit numai în I2C sclav)
1 = setează tactul
0 = ține tactul low

SSPM3:SSPM0: biții de selecție ai portului serial
0110 = I2C sclav, 7 biți de adresă
0111 = I2C sclav, 10 biți de adresă
1000 = I2C stăpân, clk = Fosc/(4*(SSPADD+1))
1011 = I2C stăpân sub control firmware, sclavul dezactivat
1110 = I2C stăpân cu control firmware, 7 biți de adresă cu start şi stop, întrerupere activă
1111 = I2C stăpân cu control firmware, 10 biți de adresă cu start şi stop, întrerupere activă
1001, 1010, 1100, 1101, rezervați SSPCON i2c

fig.6- 27 Funcțiile registrului SSPCON pentru modul I2C

```
; rutine I2C hardware pentru memorii eeprom de 8/16 biți-400KHz
; procesor PIC16F87x, utilizează biblioteca jpic
const I2C_speed = 400_000; 400KHz ; viteza de comunicație rapidă
const byte I2C_baud = ( ( target_clock / I2C_speed ) / 4 ) - 1

procedure I2C_master_init is
    pin_c4_direction = input        -- intrări
    pin_c3_direction = input
    f877_sspcon   = 0b_000_1000     -- mod master
    sspen = high                    -- i2c on
 status_rp0 = high                  -- bank1
 status_rp1 = low
 f877_sspstat = 0b_0000_0000    -- slew rate activ pentru 400KHz
 f877_sspcon2 = 0b_0000_0000    -- ack recepționat de la sclav va fi 0
 f877_sspadd  = I2C_baud        -- setează viteza de comunicație
 status_rp0 = low               -- bank0
 status_rp1 = low
end procedure

procedure i2c_start is
bank_0 SSPIF = low bank_1 -- şterge flag-ul din banc 0
SEN = high                 -- condiție de start inițiată, flag în banc 1
```

```
bank_0
while ! sspif loop end loop
end procedure

procedure i2c_restart is
bank_0 SSPIF = low bank_1  -- şterge flag-ul de întrerupere anterioară
RSEN = High                -- setează bitul de restart
bank_0
while ! sspif loop end loop   -- aşteaptă o nouă întrerupere i2c
end procedure

procedure i2c_stop is
bank_0 sspif = low bank_1  -- şterge flag-ul
pen = high                 -- condiţia de stop iniţiată
bank_0
while ! sspif loop end loop
end procedure

procedure i2c_transmit ( byte in data ) is
   sspbuf = data           -- copiază data în registrul buffer
   bank_1
   while RW loop end loop  -- aşteaptă transmisia ( R/W = 0 )
   status_c = low          -- curăţă carry
    if ACKSTAT then        -- nu s-a recepţionat ACK ?
     status_c = high       -- setează carry, eroare NoACK
    end if
   bank_0
end procedure

procedure i2c_receive ( byte out data ) is
   bank_0 SSPIF = low bank_1  -- curăţă flag-ul
   RCEN = High                -- recepţia activată
   bank_0
   while SSPIF loop end loop  -- aşteaptă terminarea recepţiei
   data = f877_SSPBUF         -- data transferată din buffer
end procedure

procedure write_eeprom ( byte in data1 ,
                         byte in data2 ,
                         byte in data3 ) is
  status_c = high
  while status_c loop      -- NoACK, transmite din nou până e OK
   i2c_restart
   i2c_transmit ( 0xa0 )   -- octet de comandă, scrie data
  end loop
  i2c_transmit ( data1 )   -- adresa MSB
  i2c_transmit ( data2 )   -- adresa LSB
  i2c_transmit ( data3 )   -- data
  i2c_stop
end procedure

procedure read_eeprom (  byte in data1 ,
                         byte in data2 ,
                         byte out data3 ) is
```

```
  status_c = high
  while status_c loop      -- NoACK,transmite din nou
   i2c_restart
   i2c_transmit ( 0xa0 )   -- octet de comandă, scrie data
  end loop
  i2c_transmit ( data1 )   -- adresa MSB
  i2c_transmit ( data2 )   -- adresa LSB
  i2c_stop
  status_c = high
  while status_c loop
   i2c_restart
   i2c_transmit ( 0xa1 )   -- octet de comandă, citeşte data
  end loop
  i2c_receive ( data3 )    -- data memorată
  i2c_stop                 -- secvență terminată
end procedure

-- main, program de test
var byte testdata
pin_b6_direction = output
pin_b7_direction = output
var byte led_rosu is pin_b7 led_rosu = off
var byte led_verde is pin_b6 led_verde = off

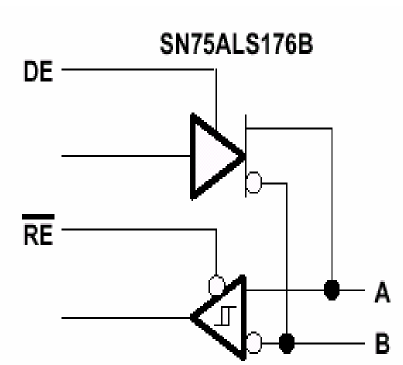
I2C_master_init ; rutina de inițializare I2C master
write_eeprom (0x00 , 0x00 , 0xAA )
read_eeprom  (0x00 , 0x00 , testdata )
 if testdata == 0xAA then
   led_verde = on
   led_rosu = off
 else
   led_rosu = on
   led_verde = off
 end if
end
```

6.3 Interfața industrială şi standardul RS485

Două din limitările majore ale interfeței RS232 sunt distanța maximă la care comunicația funcționează corect şi numărul redus (doar două) de echipamente ce poate comunica în mod half-duplex sau full-duplex pe linie. Cauza pentru care distanța este o problemă se regăseşte în modul de transmisie unipolar al semnalelor RxD şi TxD, rezistența liniei influențând nefavorabil atenuarea semnalului sub limitele impuse de marginea de zgomot a standardului. De aceea, pentru medii industriale unde zgomotele şi interferențele cu rețeaua de curent alternativ sunt mari, a fost imaginat un alt standard (EIA-485) cu imunitate mult mai bună la zgomote, funcționând cu semnale de ieşire diferențiale pe o lungime garantată a tronsonului de cablu de 1000…1200m. Avantajul suplimentar este posibilitatea interfațării unui număr mare de echipamente pe linie, 32 pentru standardul inițial sau 64 până la 256 dacă se reduce cu un factor corespunzător curentul injectat în linie (cu ½ sau ¼ din valoarea inițială standardizată de 60 mA) şi implicit lungimea liniei

[8]. Spre deosebire de EIA-232, EIA-485 nu precizează nici tipul conectorului ce trebuie utilizat pentru interfaţare şi nici protocolul software ce trebuie utilizat, ambele fiind la opţiunea utilizatorului. Ceea ce însă "este bătut în cuie" sunt limitele extreme de variaţie a semnalelor şi parţial impedanţa receptoarelor, valoarea curentului diferenţial injectat în linie şi prezenţa terminatoarelor rezistive pe linie pentru viteze de comunicaţie mari. Se observă că este un standard prietenos în care dimensionarea corectă a elementelor stă aproape în întregime pe umerii proiectantului.

Conexiunea pe RS485 se realizează pe trei fire, două din ele, notate arbitrar A şi B sunt purtătoare de semnal şi al treilea este linia de masă. Semnalele regăsite pe liniile A şi B sunt în antifază, purtătorul de informaţie fiind un curent, potenţialul corespunzător fiind măsurat faţă de linia de masă. Fiecare circuit transmiţător/receptor deţine unul sau doi pini de control a funcţiei îndeplinite (emiţător, receptor sau stare de înaltă impedanţă a ieşirii). Receptorul are o intrare diferenţială şi o ieşire de date compatibilă TTL, în timp ce transmiţătorul are intrarea compatibilă TTL şi ieşirea diferenţială . Viteza tipică garantată pe linie este de 10MB/s şi scade cu lungimea cablului. Impedanţa cablului torsadat utilizat pentru comunicaţie este de 100…120 de ohmi şi pe o singură astfel de linie torsadată se poate realiza o comunicaţie half duplex. Pentru a realiza comunicaţia full duplex e nevoie de două linii diferenţiale, cu transmiţătoarele-receptoarele aferente [9]. Standardul industrial pentru RS485 o reprezintă circuitul integrat SN75176 [10], a cărui topologie a intrărilor şi ieşirilor se regăseşte la un număr mare de circuite integrate drivere RS485, care au implementată intern şi imunitatea la descărcări electrostatice pe linie [11]. Principalele caracteristici generale ale transmiţătorului şi receptorului de RS485 se regăsesc în tabelul de mai jos:



Parametru transmiţător şi receptor RS485 Tensiunea maximă generată pe linie	Valoarea -7V…+12V
Nivel de semnal la iesire cu sarcină nominală	± 1.5V
Nivel de semnal la iesire în gol	± 6V
Impedanţa de sarcina a transmiţătorului	55…60 ohm
Gama tensiunilor de intrare în receptor	-7V…+12V
Sensibilitatea receptorului	±200mV
Impedanţa de intrare în receptor	>10Kohm

fig.6- 28 Circuitul integrat SN75176 devenit standard industrial

Liniile de control ale SN75176 sunt Data Enable şi Reception Enable. Acestea se pot conecta împreună la aceeaşi linie de control a PIC-ului, pentru a trece circuitul în recepţie sau transmisie. Caracteristica de intrare a acestui circuit este prezentată în fig. 6-29. In cazul cel mai defavorabil, (linia încărcată cu numărul maxim de emiţătoare/receptoare) şi distanţa maximă a tronsonului de cablu între emiţător şi receptor, semnalul de intrare în receptor trebuie să fie mai mare de ±200mV, receptorul având din construcţie un histerezis de cca 50mV. Limita maximă şi minimă a semnalului generat în linie este dependentă la intrarea receptorului şi de tensiunea de mod comun ce apare pe linie şi care este un parametru în funcţie de lungimea liniei şi modul de conexiune, (cu doar 2 fire active conectate la intrarile/ieşirile A şi B sau cu al treilea fir de masă între modulele emiţător şi receptor). Această tensiune de mod comun nu trebuie să depăşească valoarea -7V…+12V pentru o comunicaţie corectă. Dacă linia funcţionează în gol, tensiunea maximă pe ieşirile A şi B a transmiţătorului (măsurată faţă de masă, fig.6-28) nu va depăşi ± 6V în timp ce

tensiunea diferențială la ieşire (măsurată între ieşirile A şi B fig.6-28) trebuie să fie în limitele ± 1.5V … ±6V.

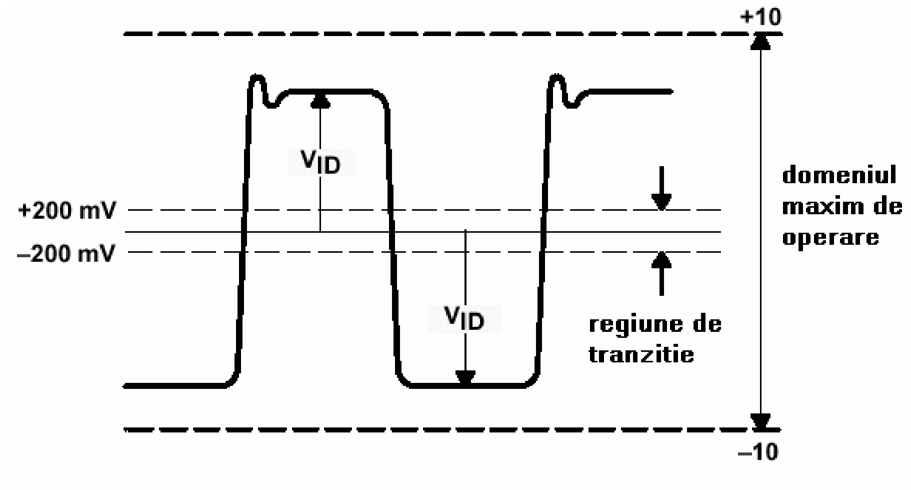


fig.6- 29 Parametrii de intrare ai receptorului ±200mV < $|V_{ID}|$ < ±10V

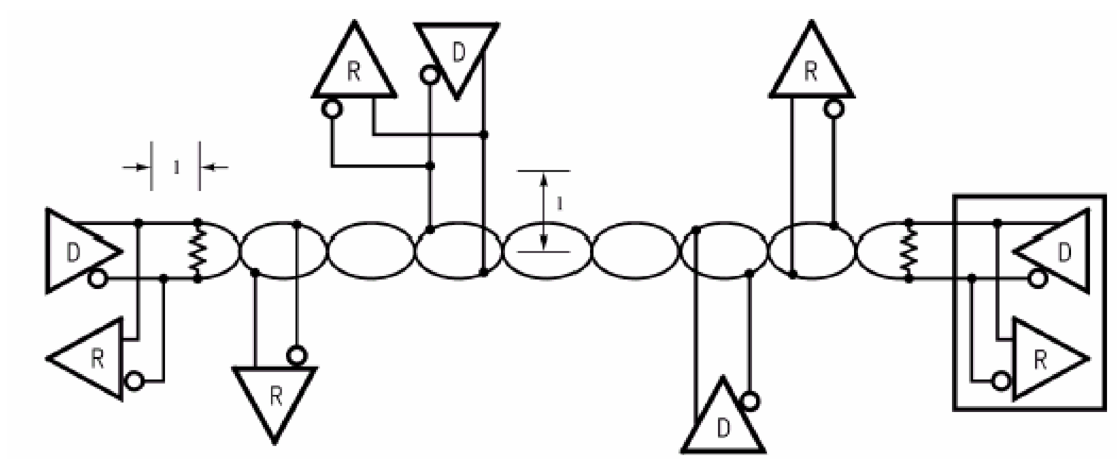


fig.6- 30 Structura posibilă a unei linii 485 cu terminatoare, D = driver, E = receiver, l = conexiune scurtă

Linia 485 din fig.6-30 utilizează terminatoare paralele rezistive având valoarea egală cu impedanța liniei (120…130 ohmi). Lungimea **l** a tronsoanelor ce conectează emițătorii/receptorii la linie va fi menținută scurtă, (maxim de ordinul zecilor de cm, în cel mai defavorabil caz 1 m). Singura topologie sigură care funcționează fără probleme este *daisy-chain* (topologie înlănțuită). Specificul acesteia este "plimbatul" cablului de comunicație (fig.6-31) de la un receptor/emițător (D/R) la celalalt, indiferent de locul unde se găsesc amplasați aceştia, astfel încât de cele mai multe ori apare o întoarcere (dublare) a cablului pe diverse tronsoane.

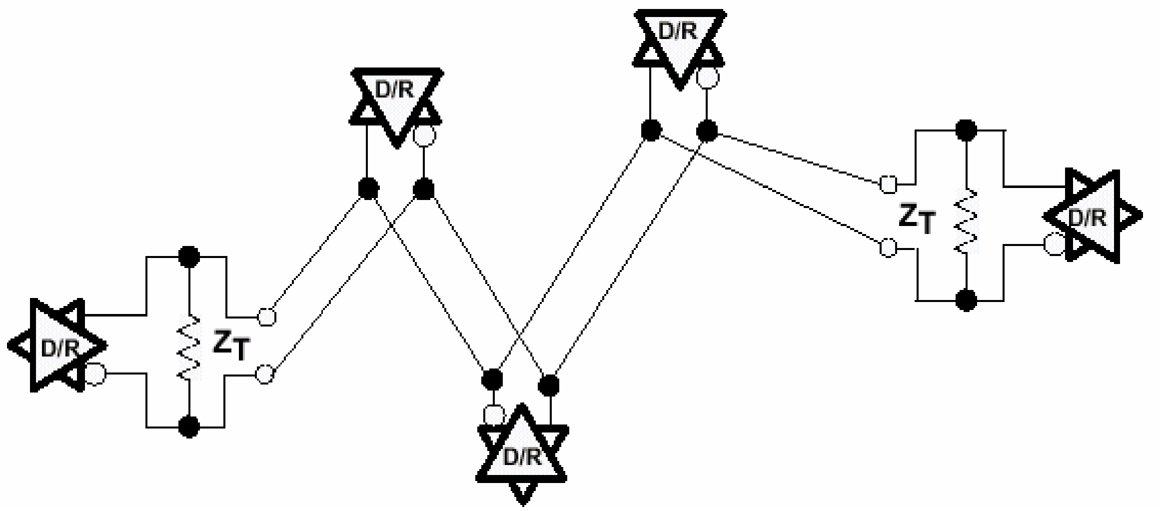


fig.6- 31 Conexiunea înlănțuită (*daisy-chain)* a transmițătoarelor/receptoarelor 485, fără linie de masă

In practică se utilizează şi terminatoare divizoare (*fail-safe*) care mențin potențialul liniei fixat, în cazul unei impedanțe ridicate globale pe linie (când toate emițatoarele/receptoarele se găsesc în stare de înaltă impedanță) (fig. 6-32). Toate driverele RS485 vor avea plantate aceste trei rezistențe pe circuitul imprimat, ele putând fi anulate prin intermediul a trei jumperi sau comutatoare superminiatură, urmând a fi activate numai pentru dispozitivele situate fizic în pozițiile terminale ale liniei. Avantajul acestor terminatoare este obținerea unui bus cu două nivele logice: *high* rezultat al unei comenzi din driver sau *high* rezultat al terminatorului *failsafe* şi *low* ca rezultat al comenzii driverului (fig.6-32). Teoria mai aminteşte de existența terminatoarelor capacitive (o rețea RC în serie conectată între

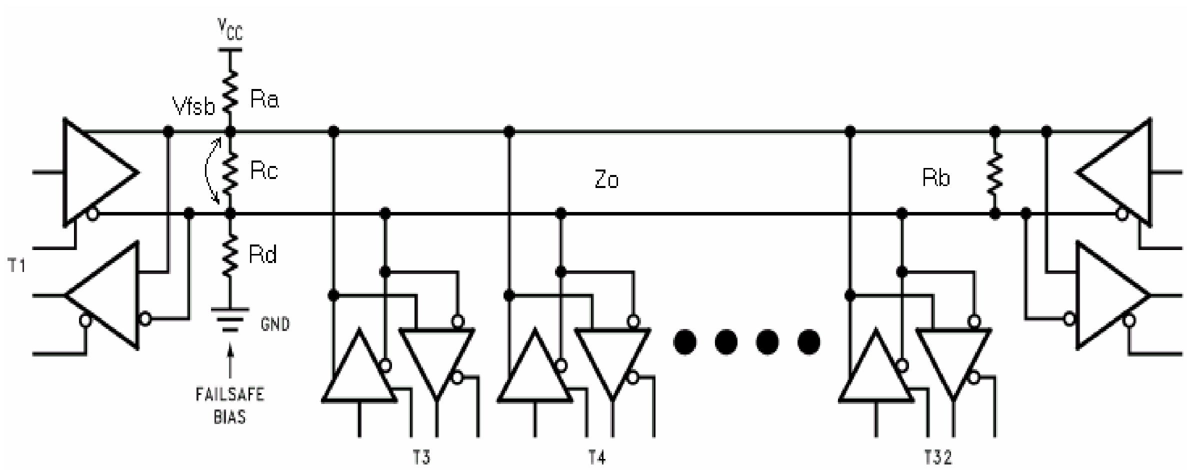


fig.6- 32 Linie RS485 cu terminator divizor.

liniile A şi B) care au avantajul de a nu fi consumatoare de curent (pe care divizorul rezistiv anterior îl consuma) şi dezavantajul scăderii vitezei de comunicație şi a lungimii cablului datorită constantei de timp proprii [12]. In practică acestea se utilizează foarte rar şi numai pentru viteze mici unde, linia poate rămâne la fel de bine şi fără terminator.

Calculul terminatorilor pentru o linie simetrică a cărei rezistență proprie se neglijează, este o problemă ce necesită doar aplicarea legii lui Ohm (fig.6-32). Impedanța cablului este Zo = 120 ohm. Presupunem că Rc şi Rb se aleg egale cu impedanța liniei iar Ra şi Rd lipsesc: Rc = Rb = Zo = 120ohm, iar rezistența echivalentă va fi:
$R_{echivalent}$ = Rc||Rb = 120/2 = 60 ohm
Cunoscând valoarea minimă necesară a Vfsb = 200mV (**V** **f**ail**s**afe **b**ias) dată de fig.6-33 şi Vcc = 5V, rezultă din teorema divizorului de tensiune:

$$Vfsb = Vcc * \frac{R_{echivalent}}{Ra + Rb + R_{echivalent}}$$

Pentru simplificare notăm Ra + Rd = 2R, pentru că linia este simetrică, şi atunci:

$$2R = \frac{Vcc}{Vfsb} * R_{echivalent} - R_{echivalent}$$

De unde, rezultă 2R = 1440 ohm sau R=Ra=Rd=720 ohm. In acest moment se poate verifica rezistența echivalentă a nodului la emisie: Rc||(Ra+Rd) = 120ohm || 1440ohm = 110ohm
Deoarece rezultatul este foarte apropiat de valoarea impedanței cablului (în limita a 10%), se poate considera că nu mai sunt necesare nici un fel de ajustări. Se poate adopta şi un calcul mai precis (însă în practică nu e necesar) după cum urmează:
Se consideră că Zo = Rc||(Ra+Rd), impedanța cablului fiind văzută la terminatorul de emisie. Pentru Zo =120 ohm şi Ra = Rd = 720 ohm calculate anterior, rezultă Rc = 131ohm. Cu această valoare $R_{echivalent}$ = Rc||Rb = 131||120 = 62ohm, valoare foarte apropiată de cea standard de 60 ohm. In acest moment se pot alege valorile cele mai apropiate ca toleranțe pentru rezistențe, după cum urmează:
Ra = Rd = 750 ohm, Rb = 120 ohm, Rc = 130ohm şi se verifică din nou condițiile :

- Rc||(Ra+Rb) = Zo 130ohm||1500ohm = 120ohm
- Rechivalent = Rb||Rc 120ohm||130ohm = 62ohm
- Vfsb ≥ 200mV, din relația de calcul prezentată mai sus

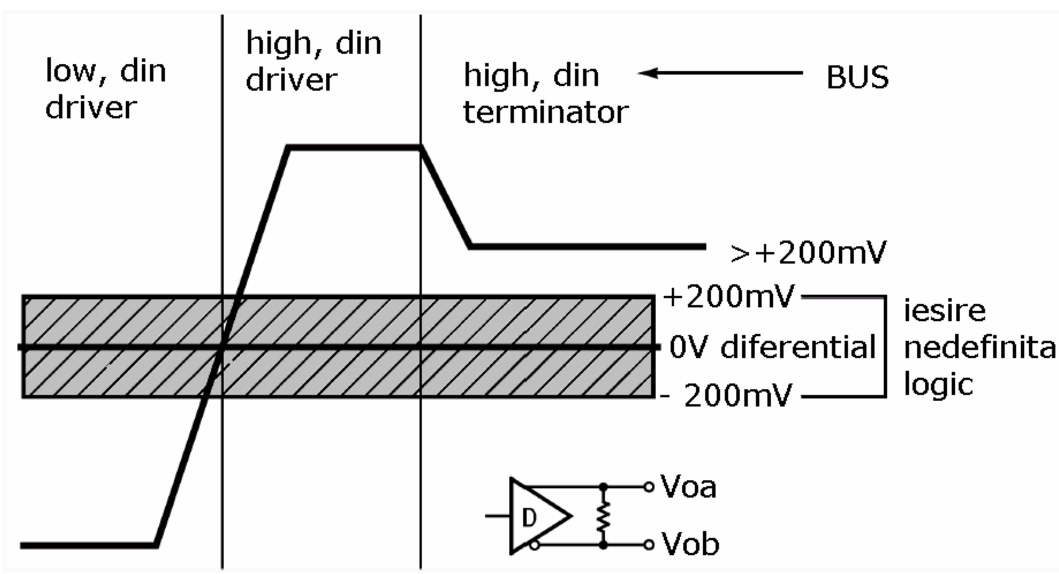


fig.6- 33 Stările logice ale BUS-ului cu terminator *failsafe*

Exemplul de calcul prezentat arată că valoarea de 750 ohmi este cea mai mare valoare admisă pentru Ra şi Rd pentru o linie şi nu se recomandă scăderea valorii acestora prea mult pentru a nu creşte consumul de curent pe linie. Dacă linia este foarte lungă şi rezistenţa proprie a cablului nu poate fi neglijată, modul de calcul al terminatoarelor este prezentat în anexa articolului [13] .

Una din precauţiile ce trebuiesc luate în cazul liniilor foarte lungi (4…5Km), este protecţia dispozitivelor la descărcări electrostatice, motiv pentru care pe lângă supresarea liniilor cu diode de protecţie la regim tranzitoriu, se utilizează şi izolarea optică a liniilor de comandă, emisie şi recepţie date şi alimentarea acestuia printr-un convertor DC-DC [14]. Bineînţeles că este nevoie de repetori la fiecare 1km, pentru a reface calitatea semnalului transportat (prin refacerea fronturilor semnalelor). Problemele de fond la realizarea acestor repetori pot fi observate în [15].

Deoarece problematica apariţiei conflictelor pe bus, a protecţiei la ESD, optoizolarea terminatoarelor, păstrarea unei tensiuni de mod comun acceptabile, repetarea semnalului, determinarea topologiei optime a liniei, a vitezei maxime de transport prin alegerea corespunzătoare a tipului de cablu, realizarea conversiei între standardul RS485 şi alte standarde de comunicaţii seriale, sunt acoperite în detaliu de materialul existent pe CD:\RS485\, voi exemplifica cum se poate implementa această comunicaţie printr-o conexiune simplă multi PIC.

6.3.1 Conexiune multi-PIC prin interfaţă EIA-485

Pentru a testa o comunicaţie via RS485, fără a avea nici un modul specializat disponibil, este imperativă proiectarea celui mai simplu convertor RS232-RS485. Acesta va asigura testarea tuturor sclavilor ce vor fi conectaţi pe bus, cu ajutorul PC-ului (programul Hyperterminal) şi a unui firmware standard de test pentru RS232, prezentat în subcapitolul 6.1. Schema convertorului este simplă :

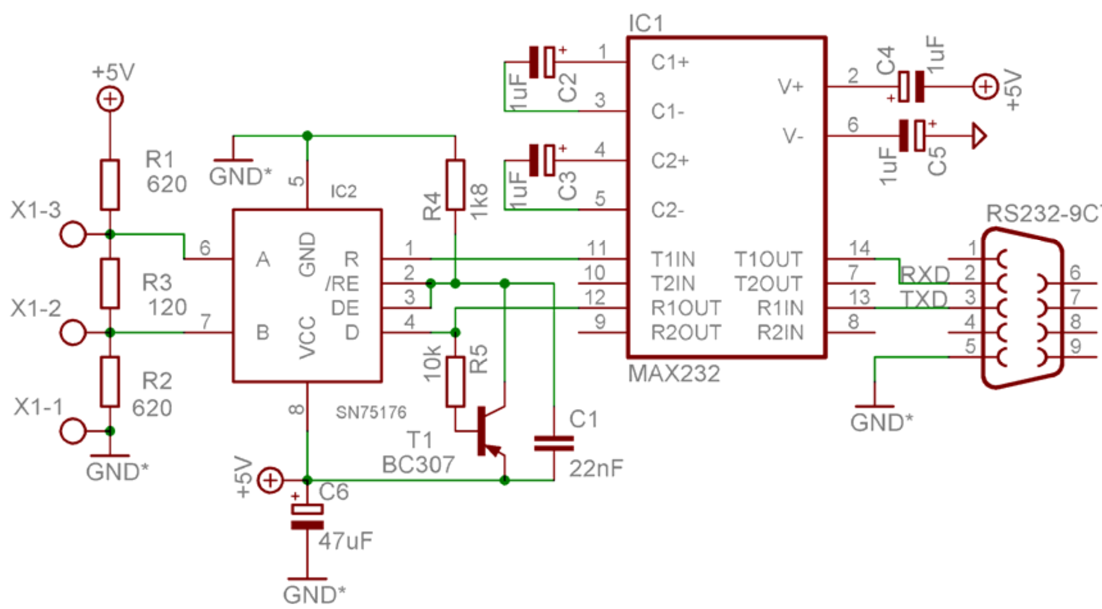


fig.6- 34 Cel mai simplu convertor RS232-RS485 half-duplex

Studiind fila de catalog a circuitului SN75176 se poate observa că /RE şi DE pot fi conectate împreună asigurând direcția de transfer a informației, conform tabelului următor:

	DE/RE = high	**DE/RE = low**
Tx	activ	HZ
Rx	HZ	activ

unde HZ reprezintă starea de înaltă impedanță, iar DE/RE conform fig.6-34 este pinul de selecție combinat al transmițătorului/receptorului RS485. Funcționarea circuitului IC1 fiind descrisă anterior, singura necunoscută rămâne metoda de trecere automată a lui IC2 din receptor în emițător RS485. Starea normală a acestuia este de receptor, DE/RE este *low* datorită rezistenței R4. Ambele semnale R şi D sunt active *low* deoarece MAX232 este inversor, deci starea liniilor TTL în repaus este *high* (standard TTL). Când potențialul liniei D trece *low* ca urmare a unui semnal activ transmis pe TXD din PC, T1 care a fost blocat întră în conducție forțând DE/RE în stare logică *high,* IC2 devenind transmițător. Este necesară o uşoară întârziere generată de R4 şi C1 pentru a menține dispozitivul în această stare şi după revenirea IC2 în regim receptor. Aceasta trebuie corelată cu viteza de transmisie (în cazul nostru maxim 9600bps) şi cu programul firmware al sclavilor conectați pe bus, deoarece structura prezentată permite doar comunicația half-duplex. Dacă această întârziere lipseşte, va apare o inadvertență materializată prin trecerea prematură a emițătorului IC2 în stare de înaltă impedanță, respectiv a convertoarelor de comunicație SN75176 ce echipează sclavii conectați pe bus. PC-ul devine *master* pentru testarea individuală a sclavilor conectați pe bus sau a modulului PIC ce va deveni *master* mai apoi în conexiunea multi-PIC. Odată testate individual aceste module, şansa de bună funcționare a comunicației între PIC-uri creşte spectaculos, evitând erorile hardware şi software combinate pentru mai mult de un modul implicat în rețea.

Master-ul (fig.6-35), utilizează modulul USART pentru comunicația RS485, având un afişaj LCD conectat pe portul B în modul 4+2 fire, aşa cum am prezentat în cap.4.2. Deoarece terminatorul *fail-safe* se găseşte montat pe cel mai îndepărtat sclav din rețea, prezența unui terminator simplu (R1) pe *master* este suficientă. Funcționarea *master*ului este de asemenea verificată cu ajutorul convertorului RS232/RS485 prezentat anterior. Firmware-ul masterului este un program terminal uşor modificat:

```
include f877_20     ; biblioteca f877 este comună pentru toate f87x
include jpic
include jdelay
include lcdp        ; biblioteca de definire a pinilor LCD
include hd447804    ; biblioteca de comunicație HD44780, 4+2
include usart       ; biblioteca USART, setată la 9600bps

 var byte data
 var byte row1_location = 0
 var byte row2_location = 0

 hd44780_clear            ; ințializarea LCD
 var bit sense is pin_c5  ; setarea direcției comunicației
 pin_c5_direction = output

forever loop
```

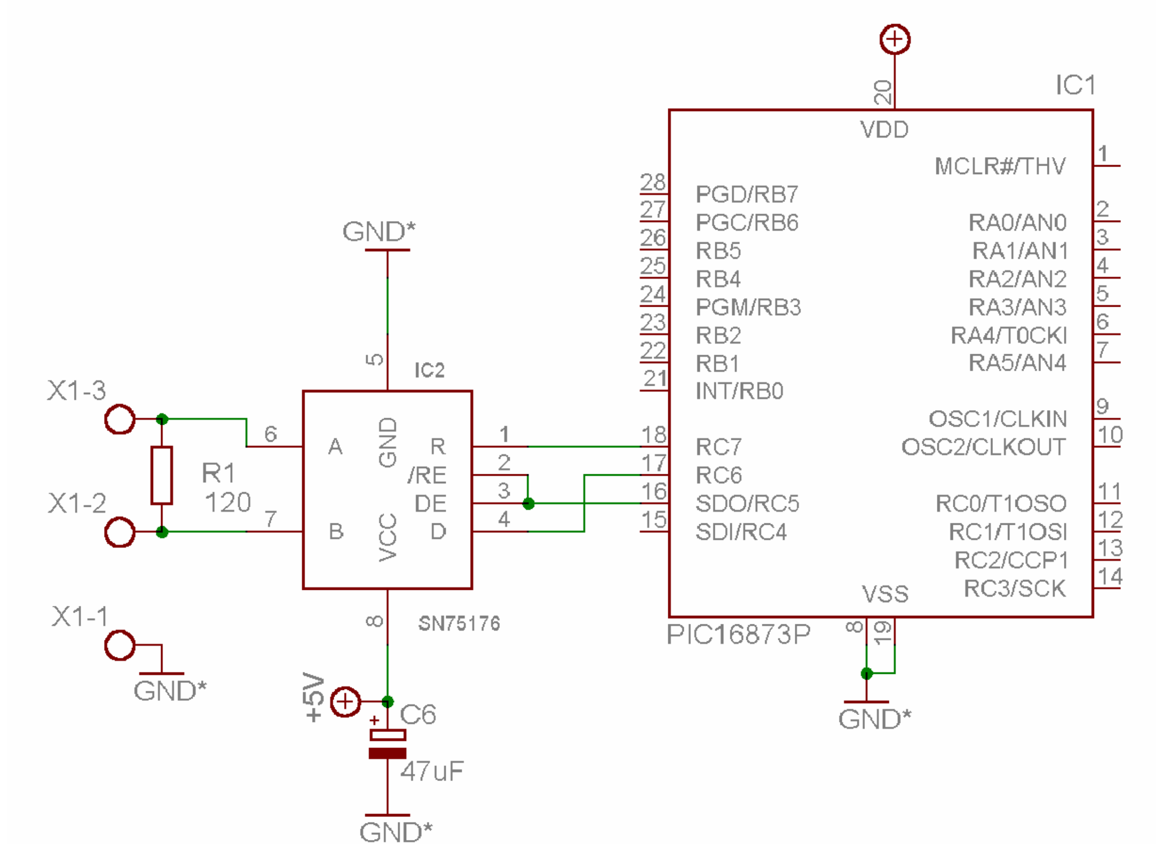


fig.6- 35 Modul RS485 master cu PIC16F873

```
  sense = low        ; 75176 receptor
  async_rx ( data )
  sense = high

 if row1_location <= 15 then
  hd44780_position1 ( row1_location )
  row1_location = row1_location + 1
  hd44780 = data
 elsif  row1_location > 15 then
  hd44780_position2 ( row2_location )
  row2_location = row2_location + 1
  hd44780 = data
   if row2_location > 16 then
    hd44780_clear
    row1_location = 0
    row2_location = 0
   end if
 end if
 sense = high
 async_tx ( data )   ; 75176 transmițător
 delay_1mS ( 5 )     ; asigură menținerea ca transmițător 5mS după
 sense = low         ; încheierea transmisiei

end loop
```

Nu am figurat elementele auxiliare modulului: oscilatorul extern, afişajul LCD ce necesită configurarea corespunzătoare a bibliotecii lcdp.jal, conectorii de alimentare, programare ICSP şi bootloader sau condensatorii de filtraj suplimentari ai PIC-ului, deoarece cititorul ştie deja cum să le conecteze din capitolele anterioare. Pentru test, *hyperterminal*-ul de sub Windows se setează la aceiaşi parametri de comunicație ca modulul USART din PIC: 9600, 8N1 fără *flow control,* comunicație pe interfața serială la care s-a conectat modulul RS232/485 realizat anterior. Tastând caractere cu hyperterminal-ul activ, acestea vor trebui să apară pe afişajul LCD şi aproape simultan pe *display*-ul calculatorului. Binențeles că trebuie executată conexiunea între liniile A/B (X1-3, X1-2 fig.6-34, fig.6-35) ale ambelor module master-slave RS485, şi dacă lungimea liniei depăşeşte 500m este bine să existe şi linia de masă, pentru minimizarea tensiunii de mod comun. Cel mai corect, linia de masă va fi torsadată cu fiecare din liniile de semnal, folosind un cablu cu cel puțin două perechi de fire. Pentru liniile foarte lungi se recomandă de asemenea conectarea masei de semnal cu masa de protecție (pământare) cu rezistențe de 100ohm în imediata apropiere a fiecărui modul.

Dacă comunicația funcționează, este momentul realizării sclavilor.

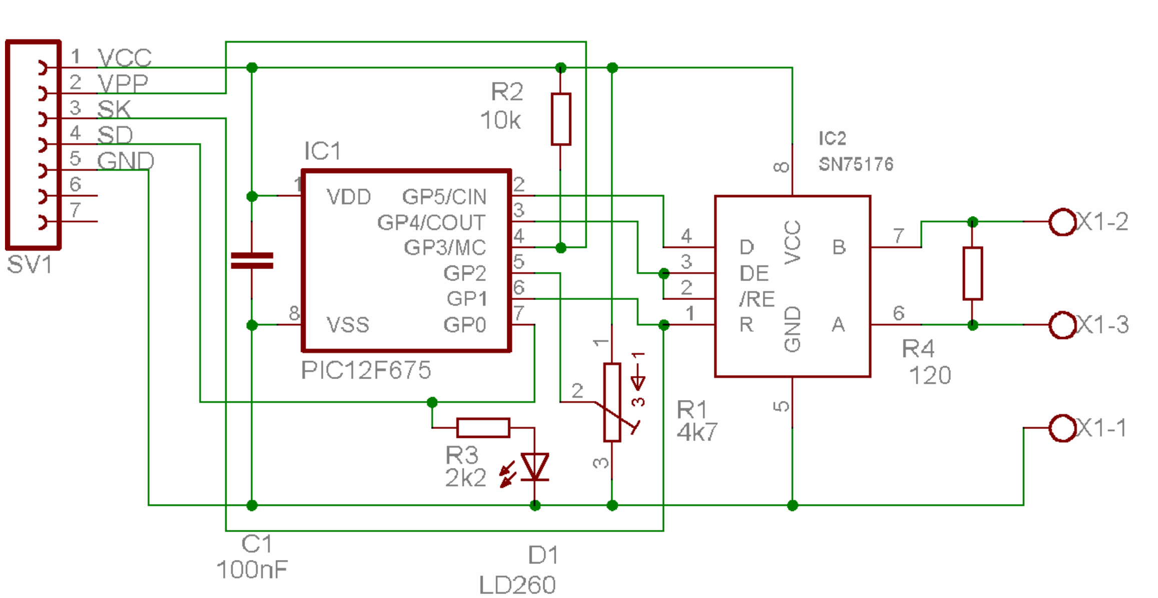


fig.6- 36 Modul sclav cu PIC12F675

Modulul din fig.6-36 este destinat citirii unei tensiuni variabile generate de cursorul potențiometrului R1 şi transmiterea acesteia pe linie prin convertorul IC2. Linia 485 nu este optoizolată iar modulul nu este protejat la descărcări electrostatice în linie, motiv pentru care schema se pretează doar transmisiilor în interiorul clădirii. Modulul dispune doar de terminator simplu (rezistența R4). Achiziționarea semnalului se face fie de un modul identic cu cel prezentat în fig.6-35, fie direct de PC (fig.6-34). Fiecare comunicație cu modulul este semnalizată optic prin intermediul diodei D1. Pe o linie clasică EIA485 (1200m/60mA) se pot conecta până la 32 de astfel de module. Viteza maximă pe linie nu va depăşi 4800 bps şi aceasta din cauza dispersiei valorilor oscilatorului intern RC al PIC12F675. Corecția frecvenței de oscilație prin software (utilizând registrul OSCCAL) este destul de neplăcută, fiind diferită pentru fiecare microcontroler în parte. Valoarea de

corecție se găseşte stocată (din fabrică) în ultima locație de memorie a PIC-ului, deci nu ştergeți PIC-ul până nu citiți locația respectivă !

Pentru realizarea software-ului sclavilor vom utiliza metoda “de la simplu la complex” din patru paşi:

1) obținerea unui program funcțional care să transmită/recepționeze un caracter înspre/dinspre sclav - PC cu viteza maximă de 4800bps, pe RS485.
2) dacă pasul 1) este funcțional, urmează citirea informației analogice de pe canalul GP2, conversia ei ASCII şi trimiterea ei în mod continuu spre terminalul PC
3) dacă pasul 2) este funcțional, vom identifica modulul cu o adresă unică (este suficientă alocarea unui singur octet pentru aceasta) şi vom condiționa trimiterea datei analogice cu recepționarea adresei valide. Rezultatul acțiunii va fi răspunsul unui şir ASCII de 4 caractere, proporționale cu tensiunea măsurată (0-1023 pentru 0-5V) după trimiterea adresei (un caracter ASCII) de la tastatura PC, sub programul hyperterminal setat corespunzător la 4800 8N1.
4) Dacă pasul 3) este funcțional, vom implementa programul ciclic de interogare al sclavilor în modulul master. Desigur ca stăpânul poate să execute şi alte operații nu numai această comunicație.

Faza inițială este configurarea corespunzătoare a tuturor bibliotecilor seriale software, eventualele diferențe între denumirea unor biți sau octeți din jpic675 şi celelalte biblioteci se corectează manual (vezi intcon_t0if şi intcon_gie).

- În biblioteca serialp.jal:

```
const bit  active_high  = true
const bit  active_low   = false ; 485/232 foloseşte un inversor !!
const byte parity_even  = 0
const byte parity_odd   = 1
const byte parity_none  = 2

const asynch_baudrate = 4800 ; maximum obtenabil cu osc. intern
const asynch_polarity = active_low ; pentru RS232/485 fig.6-34
const asynch_parity   = parity_none
const asynch_stopbits = 1

var volatile bit asynch_in_pin        is gp1
var volatile bit asynch_in_direction  is gp1_direction

var volatile bit asynch_out_pin       is gp5
var volatile bit asynch_out_direction is gp5_direction
; pinii de comunicație sunt conform cu fig.6-36
```

- În biblioteca 12F675_4I sau f675_4I trebuie configurat corespunzător oscilatorul intern conform cuvântului de configurare (vezi fila de catalog, capitolul *Special Features of the CPU*)

```
pragma target fuses 0x1184 ; sau
; pragma target fuses 0b_01_0001_0100_0100
```

Acest cuvânt este organizat pe 13 biți şi este scris în faza inițială de programare. Majoritatea programatoarelor recunosc codul corespunzător în fila hexa, dar modificarea biților poate fi

făcută şi manual din software-ul programatorului. Semnificația R/P din tabelul următor este **R**ead/**P**rogram, bit ce poate fi citit sau programat.

BG1	**BG0**	**-**	**-**	**-**	**/CPD**	**/CP**
13R/P	12R/P				8R/P	7R/P

BODEN	**MCLRE**	**/PWRTE**	**WDTE**	**FOSC2**	**FOSC1**	**FOSC0**
6R/P	5R/P	4R/P	3R/P	2R/P	1R/P	0R/P

BG1:BG0: sunt biții de calibrare ai referinței interne de tensiune
00 = tensiune redusă , 01 tensiune mărită, vezi jpic675
/CPD: bitul de protecție al memoriei de date
1 = protecția dezactivată, 0 = protecția activată
/**CP:** bitul de protecție al memoriei program
1 = protecția dezactivată, 0 = protecția activată
BODEN: bitul de setare al detecției de tensiune scăzută pe alimentare
1 = activat, 0 = dezactivat
MCLRE: bitul de selecție al modului GP3/MCLR
1 = GP3/MCLR este MCLR, 0 = GP3/MCLR este IO, MCLR este conectat intern la Vcc
/PWRTE: bitul de setare al timerului de siguranță la aplicarea alimentării
1 = PWRT este dezactivat, 0 = PWRT activat
WDTE: bitul de activare al câinelui de pază
1 = WDT activat, 0 = WDT dezactivat
FOSC2:FOSC1: biții de selecție ai oscilatorului
111 = oscilator extern RC, GP4 este CLKOUT, RC se conectează pe GP5
110 = oscilator extern RC, GP4 este pin IO, RC se conectează pe GP5
101 = oscilator intern, GP4 este CLKOUT, GP5 este pin IO
100 = oscilator intern, GP4 şi GP5 sunt pini IO,
011 = tact extern, GP4 este pin IO, tactul se aplică pe GP5
010 = oscilator extern HS, cuarțul/rezonatorul de mare frecvență (>10MHz) pe GP4,GP5
001 = oscilator extern XT, cuarțul/rezonatorul pe pinii GP4,GP5
000 = oscilator extern LP, cuarțul de max 200KHz pe pinii GP4,GP5

fig.6- 37 Cuvântul de configurare pentru PIC12F675

Programul ce testează sclavul (**pasul 1**) este următorul:

```
                    ; program de test pas1
include 12f675_4I ; biblioteca de definire a PIC-ului
include jpic675   ; aici sunt definite resursele PIC
include serialp   ; bibliotecă de configurare a pinilor transmisiei
include seriali   ; conține rutinele seriale

osc_calibrate     ; calibararea oscilatorului intern pe 4.0MHz
disable_ad        ; dezactivează convertorul AD
cmcon = 7          ; şi comparatorul intern !

var bit sense is gp4       ; sensul transmisiei half duplex
```

```
gp4_direction = output
var bit led is gp0
gp0_direction = output led = off ; stinge LED-ul
const bit receive = low
const bit transmit = high
var byte data
forever loop
       sense = receive     ; receptor 485
       asynch_receive ( data )
       sense = transmit    ; emițător 485
       asynch_send ( data )
       led = on
       delay_10mS ( 1 )    ; afişează minim 10mS ca să vedem ceva
       led = off
end loop
```

Programul va funcționa din prima încercare dacă se respectă condițiile impuse anterior. Realizarea **pasului 2** este acum mult mai simplă:

; program ce transferă terminalului valoarea tensiunii pe pinul GP2 în format ASCII

```
include 12f675_4i
include jpic675
include serialp
include seriali
include bin2bcd3    ; rutină de conversie binar/bcd împachetat
include jascii      ; bibliotecă de caractere ASCII nestandard

osc_calibrate       ; calibrează oscilatorul RC intern
disable_ad          ; toți pinii sunt IO digitale
cmcon = 7           ; comparatorul este dezactivat
var bit sense is gp4
gp4_direction = output
var bit led is gp0
gp0_direction = output
led = off
const bit receive = low
var byte adr_lo, msd, isd, lsd, ascii1, ascii2, ascii3, ascii4

forever loop
 one_ch_ad_read ( 2, ref_vdd, right_justify )
           ; rezultat in ADRESH ( bank0 ) şi ADRESL ( bank1 )
    bank_1
    asm movf adresl,w
    bank_0
    asm movwf adr_lo
 bin2bcd3 ( msd, isd, lsd, adresh, adr_lo )
 ascii1 = ( isd >> 4 )  + 48     ; mii
 ascii2 = ( isd & 0x0F ) + 48    ; sute
 ascii3 = ( lsd >> 4 ) + 48      ; zeci
 ascii4 = ( lsd & 0x0F ) + 48    ; unități
sense = transmit
    asynch_send ( ascii1 )
    asynch_send ( ascii2 )
    asynch_send ( ascii3 )
```

```
    asynch_send ( ascii4 )
    asynch_send ( ascii_cr ) ; carriage return
end loop
```

In programul anterior am utilizat un algoritm matematic pentru conversia rezultatului BCD în ASCII. Verificarea acestei conversii a fost făcută printr-un mic program, cu ajutorul simulatorului matematic intern al compilatorului:

```
include f675_4i
include jpic675
include bin2bcd3

var byte msd, isd, lsd, ascii1, ascii2, ascii3, ascii4
bin2bcd3 ( msd, isd, lsd, 0b_0000_0011, 0b_1111_1111 )

pragma test assert isd == 0b_0001_0000
pragma test assert lsd == 0b_0010_0011

 ascii1 = ( isd >> 4 )  + 48 ; mii
 ascii2 = ( isd & 0x0F ) + 48 ; sute
 ascii3 = ( lsd >> 4 ) + 48  ; zeci
 ascii4 = ( lsd & 0x0F ) + 48  ; unitati

pragma test assert ascii1 == "1"
pragma test assert ascii2 == "0"
pragma test assert ascii3 == "2"
pragma test assert ascii4 == "3"
pragma test done
```

Rezultatul conversiei binar-bcd este un format BCD împachetat. Lsd şi isd conțin fiecare câte două cifre, msd este zero în permanență deoarece numărul maxim convertit este reprezentat doar pe 10 biți (0b_11_1111_1111 = 0d_1023). Pentru această valoare, isd = 10 iar lsd = 23. Separarea celor două cifre BCD din pachet se face cu o rotire la dreapta de 4 ori (se obțin cifrele 1 din isd respectiv 2 din lsd) iar o mascare cu 0x0F a valorii inițiale duce la obținerea cifrelor 0 din isd respectiv 3 din lsd. Conversia suplimentară ASCII este necesară pentru compatibilizarea afişării cu programul terminal al PC şi necesită o adăugare de offset egală cu valoarea "0" adica 0d_48.

Modificarea programului pentru **pasul 3** devine acum o problemă banală, este nevoie doar de condiționarea transmisiei la recepția corectă a adresei:

```
var volatile byte device_address = 32 ; ascii_space
var byte data
; celelate variabile se definesc conform cu exemplul anterior

forever loop
 sense = receive
 asynch_receive ( data )
 if data == device_address then
       one_ch_ad_read ( 2, ref_vdd, right_justify )
            ; rezultat in ADRESH ( bank0 ) şi ADRESL ( bank1 )
       bank_1
       asm movf adresl,w
```

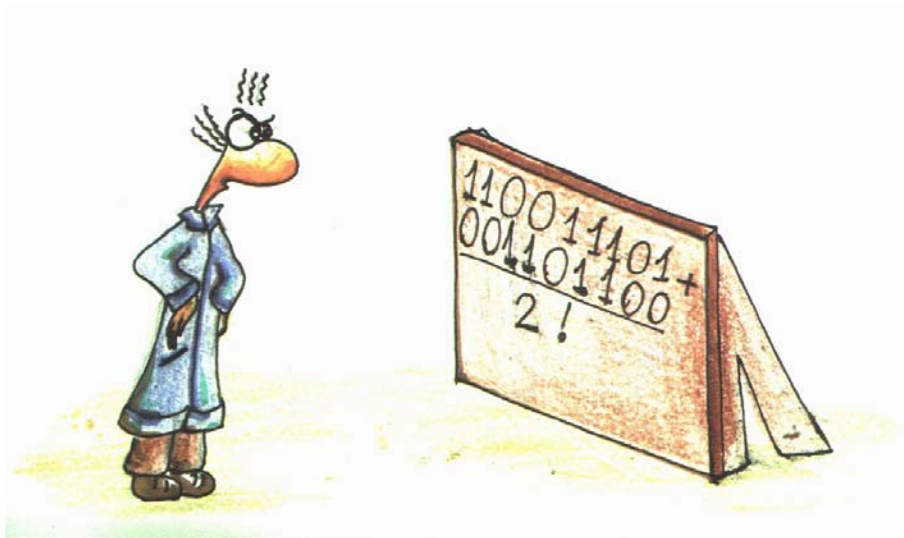
```
        bank_0
        asm movwf adr_lo
         bin2bcd3 ( msd, isd, lsd, adresh, adr_lo )
         ascii1 = ( isd >> 4 )  + 48      ; mii
         ascii2 = ( isd & 0x0F ) + 48     ; sute
         ascii3 = ( lsd >> 4 ) + 48       ; zeci
         ascii4 = ( lsd & 0x0F ) + 48     ; unități
         sense = transmit
          asynch_send ( ascii1 )
          asynch_send ( ascii2 )
          asynch_send ( ascii3 )
          asynch_send ( ascii4 )
          asynch_send ( ascii_cr ) ; carriage return
 else return
 end if
end loop
```

Fiecare adresare a sclavului prin apăsarea tastei SPACE a PC-ului, va returna o citire a convertorului AD al PIC12F675 sub acelaşi program terminal. In acest moment putem spune că problema este rezolvată integral, implementarea interogării în master fiind suficient de simplă ca să nu creeze probleme. Trebuie totuşi să avem în vedere că semnalele de comunicație vor fi active *high*, nu mai avem circuitul MAX232 în circuit, şi biblioteca serialp trebuie setată în concordanță cu realitatea (*active high*). Deoarece nu mai suntem condiționați de calculator, datele pot fi trimise atât în format ASCII cât şi în format binar. Lăsăm plăcerea cititorului să realizeze singur progrămelul care va soluționa **pasul4**.

Bibliografie:

1. Specificații RS232, http://www.camiresearch.com/Data_Com_Basics/RS232.standard.html
2. Mike Predko, RS232 The Non-Standard, http://www.mike.com
3. PIC16F87x - datasheet, DS30292A DS30292B DS30292C, 1998-2001, Microchip Technology Inc.
4. USART.asm - Thomas McGahee, http://www.piclist.com
5. Protocolul I2C, http://www.esacademy.com/index.htm
6. 24C04 datasheet, Microchip Technology Inc.
7. 24LC21 datasheet, Microchip Technology Inc.
8. Application Report SLLA112, march2002, RS485 for E-meter application, Texas Instruments
9. 485 Burr Brown application note, octomber 1997
10. SN75176b datasheet, Texas Instruments, SLLS101b, june 1999
11. Bob Perrin, RS485 Art and Science, Circuit Cellar online, http://www.circuitcellar.com
12. Application Report SLLA070C, june 2002 , RS485\422 and 485 Standards Overview and System Configuration, Texas Instruments
13. Application Note 847, july 1992, Failsafe Biassing of Differential Busses, National Semiconductor
14. Interface circuits for TIA-EIA-485, SLLA036, november 1998, Texas instruments,
15. Application Note 702, may 1990, Build a Direction Sensing Bidirectional Repeater, National Semiconductor

7 Algoritmi şi formate numerice



Algoritmii numerici reprezintă cheia prelucrării informaţiei ce intră sau iese din sistemul cu microcontroler. Deoarece există o infinitate de formate în care semnalele digitale sunt livrate pentru a fi cititite, prelucrate şi afişate, există o infinitate de soluţii pentru rezolvarea problemei. Niciodată modul de implementare al algoritmului în PIC nu va fi singular. Capitolul 7 conţine o colecţie de rutine scrise sau doar utilizate de autor în diverse experimente ale acestuia. Este probabil ca multe din problemele prezentate să aibă şi o altă soluţie mult mai bună. Adică, fie mai scurtă din punctul de vedere al codului hexagesimal generat, fie mai rapidă.

7.1 Formate numerice

Având la dispoziţie un număr mare de biţi şi octeţi, reprezentarea informaţiei cu aceştia poate lua practic orice formă. Pentru a putea interfaţa diverse sisteme logice, s-au imaginat standarde care să faciliteze transferul datelor între echipamente dezvoltate de N producători diferiţi. Câteva din formatele numerice întâlnite de utilizatorul de microcontrolere sunt prezentate în continuare.

7.1.1 Complement faţă de 2

Acest format se utilizează pentru a reprezenta atât numerele pozitive cât şi cele negative. Numărul poate avea orice mărime (ca număr de biţi), şi poate fi de tip întreg sau zecimal. Pentru un octet, reprezentarea sa este următoarea:

binar	zecimal
0xxxxxxx	0…+127
1xxxxxxx	0…-127

tabel 7- 1 Complement faţă de 2 pe 8 biţi

Valoarea numărului este reprezentată pe 7 biţi iar semnul acesteia, de bitul cel mai semnificativ. Există şi o altă posibilitate de reprezentare a unei informaţii în complement faţă de 2, de exemplu 12 biţi, reprezentaţi pe doi octeţi:

binar	**zecimal**
00000xxx_xxxxxxxx	0…+2047
11111xxx_xxxxxxxx	0 …-2047

tabel 7- 2 Complement faţă de 2 pe 16 biţi

In această situaţie, primii 5 biţi sunt utilizaţi pentru memorarea semnului, în timp ce 2048 stări distincte ale celor 11 biţi rămaşi, sunt purtătoare de informaţie. Este cazul unor convertoare AD, sau a unor senzori inteligenţi de temperatură ca DS18B20 (Dallas) sau TC77 (Microchip). Avantajul constă în simplitatea metodei de extracţie a informaţiei de semn, fiind necesară analiza acelui bit de semn din cei 5, care este cel mai uşor de manipulat (prin rotire, comparare, etc.).

7.1.2 BCD şi BCD împachetat

Formatul BCD se utilizează de obicei la pregătirea informaţiei pentru decodare utilizând decodoare standard BCD/7segmente. Spre deosebire de codul hexagesimal standard, codul BCD utilizează pentru reprezentare numai numerele de la 0 la 9:

nibble	**bcd**	**zecimal/hexa**
0000	0000	0
0001	0001	1
0010	0010	2
0011	0011	3
0100	0100	4
0101	0101	5
0110	0110	6
0111	0111	7
1000	1000	8
1001	1001	9
1010	-	A
1011	-	B
1100	-	C
1101	-	D
1110	-	E
1111	-	F

tabel 7- 3 Reprezentarea unui nibble în format BCD şi (hexa)zecimal

Deoarece numărul BCD nu poate ocupa decât un *nibble* iar microcontrolerul operează cu octeți, este frecvent utilizată împachetarea a două cuvinte BCD într-un octet. Astfel se utilizează mult mai bine capacitatea fiecărui octet, cu efecte benefice în salvarea unor regiştrii de uz general din microcontroler, necesari în cazul programelor tip mamut (cu dimensiune mare şi consumatoare de regiştrii SRAM). Separarea cuvintelor BCD se poate face prin mascare cu 0FH pentru obținerea cuvântului BCD mai puțin semnificativ (LBCD), sau prin mascare cu F0H urmată de o rotație completă spre dreapta de 4 poziții sau urmată de procedura *swap_nibble*, pentru cuvântul BCD mai semnificativ (MBCD). Explicația anterioară este aplicată în practică în cap.6.3.1.

MBCD	**LBCD**
xxxx	xxxx

tabel 7- 4 Formatul BCD împachetat

7.1.3 Codul ASCII

Codul ASCII este o reprezentare standardizată a caracterelor tipăribile şi a unor comenzi. Apăsând un caracter pe tastatura calculatorului dvs., generați spre unitatea centrală transmisia codului ASCII corespunzător. In cap. 2.9.10 sunt prezentate caracterele ASCII care nu pot fi tipărite, conținute într-o bibliotecă specifică Jal. O reprezentare unitară a setului de caractere ASCII se găseşte în tabelul următor:

		MSC							
	hex	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
L	**0**	NUL	DLE	Space	0	@	P	`	p
S	**1**	SOH	DC1	!	1	A	Q	a	q
C	**2**	STX	DC2	"	2	B	R	b	r
	3	ETX	DC3	#	3	C	S	c	s
	4	EOT	DC4	$	4	D	T	d	t
	5	ENQ	NAK	%	5	E	U	e	u
	6	ACK	SYN	&	6	F	V	f	v
	7	Bell	ETB	'	7	G	W	g	w
	8	BS	CAN	(	8	H	X	h	x
	9	HT	EM	)	9	I	Y	i	y
	A	LF	SUB	*	:	J	Z	j	z
	B	VT	ESC	+	;	K	[	k	{
	C	FF	FS	,	<	L	\	l	|
	D	CR	GS		=	M	]	m	}
	E	SO	RS	.	>	N	^	n	~
	F	SI	US	/	?	O	_	o	DEL

tabel 7- 5 Codul ASCII, combinație de MSC, LSC

Cuvântul ASCII se formează ca o combinaţie a **M**ost **S**ignificat **C**haracter (caracterul cel mai semnificativ) respectiv a **L**ast **S**ignificant **C**haracter (caracterul cel mai puţin semnificativ). De exemplu, caracterul ASCII "A" = 41h = 65d = 0100_0001b.

Transmisia informaţiei prin caractere ASCII este mai lentă decât transmisia prin cod binar (sunt necesare cel puţin 2 caractere ASCII neîmpachetate pentru fiecare octet de transmis), însă avantajul major este compatibilitatea cu orice sistem de calcul ce utilizează setul ASCII, inclusiv dispozitivele periferice de tipărire sau terminalele cu putere de calcul modestă. Inclusiv aparatele de măsură digitale direct interfaţabile cu PC-ul, folosesc cu precădere acest set standardizat pentru transmisia informaţiei. Utilizatorul îşi poate imagina cu uşurinţă şi o modalitate de compresie a caracterelor ASCII transmise, deoarece valoarea maximă este mai mică decât 7Fh (127d), ceea ce permite gruparea a doi ASCII pe un octet (FFh = 255d).

7.1.4 Formatul zecimal cu virgulă mobilă (floating point)

In formatul zecimal cu virgulă mobilă [3], numărul este reprezentat ca o expresie de forma:

$$x = \text{mantisa} * 2^{\text{exponent}}$$

Numărul biţilor conţinut de mantisă determină precizia de exprimare a numărului în timp ce numărul de biţi ai exponentului determină domeniul de mărime al numărului reprezentat. Acest format poate avea o infinitate de valori atât pentru mantisă cât şi pentru exponent. Ca exemplu, numărul zecimal 10 poate fi reprezentat în următoarele moduri:

$10 * 2^0$; $5 * 2^1$; $2.5 * 2^2$; $1.25 * 2^3$; $0.625 * 2^4$; $0.3125 * 2^5$

Reprezentarea binară a numerelor fracţionare este identică cu cea a numerelor întregi, singura deosebire este faptul că zecimea reprezintă ½ din bit, sutimea ¼ din bit, miimea 1/8 din bit s.a.m.d. Conversia mantisei de mai sus în cod binar are forma următoare:

10 = 1010
5 = 101
2.5 = 10.1
1.25 = 1.01
0.625 = **0.1**01
0.3125 = 0.0101

Există şi o regulă pentru a alege combinaţia potrivită de mantisă/exponent din infinitatea de posibilităţi: acea mantisă este validă care are 0 în partea stângă a virgulei şi 1 în partea dreaptă a virgulei, pentru cazul de mai sus este $0.625 * 2^4$. Toate celelalte valori se "normalizează" până se aduc la această formă. Dacă dorim să o convertim în format binar, vom avea mantisa = 0.101 (0.625d) iar exponentul = 100 (4d).

Deoarece formatul valid al mantisei este întotdeauna 0.1xxx, partea care rămâne constanta (0.1), poate fi omisă din reprezentare. In locul ei se poate instala bitul de semn (MSB al mantisei) care este 1 pentru numere pozitive si 0 pentru numere negative. Astfel mantisa numărului nostru devine 01, iar dacă introducem şi semnul va fi 101. Lucrurile ar

rămâne simple dacă nu s-ar efectua şi o "ajustare" a exponentului dependentă de numărul de biți reprezentați, conform relației:

Exponent_ajustat = exponent + 2^{n-1} unde: n este numărul de biți al reprezentării

Astfel pentru o mantisă de 16 biți şi un exponent de 8 biți (o reprezentare posibilă în PIC), numărul nostru va fi:
+10 =>   mantisa = 1010_0000_0000b   exponent = 100 + 1000_0000 (2^7) = 1000_0100b
In mod asemănător se poate deduce valoarea lui –10, în acelaşi format cu virgula flotantă de 16 biți mantisa, 8 biți exponentul:
-10 =>   mantisa = 0010_0000_0000b   exponent = 1000_0100b

Inmulțirea şi împărțirea cu virgulă flotantă respectă relațiile matematice clasice:

Dacă $x = mantisa_x * 2^{exponent_x}$ şi $y = mantisa_y * 2^{exponent_y}$
atunci:

$$x \times y = mantisa_x \times mantisa_y \times 2^{(exponent_x + exponent_y)}$$

respectiv:

$$\frac{x}{y} = \frac{mantisa_x}{mantisa_y} \times 2^{(exponent_x - exponent_y)}$$

Adunarea şi scăderea implică efectuarea unor operații distincte:

1. Compararea exponeților prin scădere şi determinarea celui mai mare
2. Alinierea mantiselor, mantisa cu exponentul cel mai mic se roteşte spre dreapta cu un număr de ori egal cu rezultatul scăderii anterioare
3. Adunarea mantiselor aliniate, rezultatul operației primeşte exponetul cel mai mare al operanzilor
4. Normalizarea mantisei rezultatului
5. Ajustarea exponentului

Utilizarea bibliotecii matematice cu virgulă flotantă este în general dificilă, fiindcă utilizarea corectă a mantisei şi a exponentului cade exclusiv în sarcina utilizatorului. Există mai multe formate cu virgulă flotantă acceptate de PIC, frecvent utilizate sunt formatul Microchip pe 24 de biți:

EEEE_EEEE_SMMM_MMMM_MMMM_MMMM

Microchip pe 32 de biți:

EEEE_EEEE_SMMM_MMMM_MMMM_MMMM_MMMM_MMMM

Sau formatul IEE754 pe 32 de biți:

SEEE_EEEE_EMMM_MMMM_MMMM_MMMM_MMMM_MMMM

Unii creatori de compilatoare pentru microcontrolere PIC şi-au impus propriul format, ca de exemplu formatul Hitech pe 24 de biți:

SEEE_EEEE_EMMM_MMMM_MMMM_MMMM

In formatele de mai sus S reperezintă semnul, E reprezintă exponentul ajustat cu –127d iar M, mantisa.

7.1.5 Formatul zecimal cu virgulă fixă (fixed point)

Formatul zecimal cu virgulă fixă utilizează un număr de biţi pentru numărul întreg, urmat de un alt număr de biţi pentru partea zecimală a numărului. Poate reprezenta numere pozitive sau numere pozitive şi negative dacă partea întreagă este figurată în complement faţă de 2. Simbolizarea reprezentării binare cu virgulă fixă pentru un caz particular, este:

xxxxxxx.xxx		7Q3
n	m	nQm

tabel 7- 6 Formatul cu virgulă fixă

Aceasta înseamnă că sunt utilizaţi **n** biţi pentru reprezentarea părţii întregi şi **m** biţi pentru partea fracţionară. Tabelul următor exemplifică acest mod de reprezentare în binar:

întreg	fracţionar
10000000 = 128	.10000000 = 1/2 (128/256) .5
01000000 = 64	.01000000 = 1/4 (64/256) .25
00100000 = 32	.00100000 = 1/8 (32/256) .125
00010000 = 16	.00010000 = 1/16 (16/256) .0625
00001000 = 8	.00001000 = 1/32 (8/256) .03125
00000100 = 4	.00000100 = 1/64 (4/256) .015625
00000010 = 2	.00000010 = 1/128 (2/256) .0078125
00000001 = 1	.00000001 = 1/256 (1/256) .00390625

tabel 7- 7 Reprezentarea zecimală a unor numere binare întregi şi fracţionare

Orice număr binar cu virgulă fixă poate fi scris ca o sumă de termeni conţinuţi în tabelul 7-7. Este evident că rezoluţia mărimii reprezentate este dependentă de numărul de biţi, şi că aceasta poate fi diferită pentru partea întreagă, respectiv pentru cea zecimală. Adunările şi scăderile se efectuează în mod distinct, separat cu partea întreagă şi separat cu cea zecimală, depăşirea unităţii în partea zecimală generând un *carry* (depăşire) sau un *borrow* (împrumut) care se aplică părţii întregi (depăşirea se adună părţii întregi respectiv împrumutul se scade din partea întreagă).

Cel mai bun exemplu este modul de codare cu virgulă fixă a informaţiei obţinute prin citirea senzorului de temperatură DS18B20. Acesta seamănă ca două picături cu DS18S20, senzor prezentat în cap.4.12, singura deosebire evidentă este modul de împachetare al datelor de ieşire pe doi octeţi, (LSByte este octetul cel mai puţin semnificativ, MSByte este octetul cel mai semnificativ, LSBit este bitul 0 iar MSBit este bitul 7 pentru cei doi octeţi), dependenţa de temperatură fiind cea expusă în tabelul 7-8:

2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	LSByte
MSBit			Unitatea = °C				LSBit	
S	S	S	S	S	2^6	2^5	2^4	MSByte

tabel 7- 8 Reprezentarea temperaturii în DS18B20

Temperatură °C	Cod binar		
	semn	**întreg**	**zecime**
+125	0000_0111_1101_**0000**		
+85	0000_0101_0101_**0000**		
+25.0625	0000_0001_1001_**0001**		
+10.125	0000_0000_1010_**0010**		
+0.5	0000_0000_0000_**1000**		
0	0000_0000_0000_**0000**		
-0.5	1111_1111_1111_**1000**		
-10.125	1111_1111_0101_**1110**		
-25.0625	1111_1110_0110_**1111**		
-55	1111_1100_1001_**0000**		

tabel 7- 9 Dependenţa codului binar generat de temperatura reală, DS18B20

Detaliind tabelul, observăm că pentru rezoluţia maximă a părţii zecimale sunt utilizaţi 4 biţi, rezoluţiile absolute corespunzând celor 16 stări distincte ale acestora (pentru temperaturi pozitive), fiind următoarele:

0000	**0001**	**0010**	**0011**	**0100**	**0101**	**0110**	**0111**
0	0.0625	0.125	0.1875	0.25	0.3125	0.375	0.4375
1000	**1001**	**1010**	**1011**	**1100**	**1101**	**1110**	**1111**
0.5	0.5625	0.6250	0.6875	0.75	0.8125	0.8750	0.9375

tabel 7- 10 Rezoluţia părţii zecimale a temperaturii, pentru DS18B20

7.2 Conversii ale diferitelor formate

7.2.1 Conversia unei mărimi prin metoda comparării cu momente de referinţă fixe (metoda tabelului de conversie)

Dacă continuăm analiza tabelului 7-8, observăm că avem nevoie de conversia valorii ieşirii (temperatură) pentru mărimea de intrare (biţi), atât pentru reprezentarea întreagă cât şi pentru reprezentarea zecimală a temperaturii. Metoda cea mai simplă este evidenţiată de programul următor care utilizează algoritmul de comparare cu momente de referinţă fixe ale mărimii de intrare:

```
var byte dec, decimal, units, tens, temperature
forever loop
```

```
   DS1820_start_temperature_conversion
; porneşte conversia de temperatură a DS18B20
   DS1820_read_temperature_raw ( msb, lsb )
; citeşte cei doi octeţi ce stochează temperatura
   comp2_binary
; msb este convertit din complement faţa de 2 în binar
   dec = lsb & 0x0F     ; extrage partea zecimală
   units = lsb & 0xF0
; extrage partea întreagă prin mascare
   units = units >> 4   ; şi rotire la poziţia corectă
; începe compararea conform tabelului 7-9
   if    dec ==  0 then decimal =  0
   elsif dec ==  1 then decimal =  6      ; 0.0625
   elsif dec ==  2 then decimal = 12      ; 0.125
   elsif dec ==  3 then decimal = 18      ; 0.1875
   elsif dec ==  4 then decimal = 25      ; 0.25
   elsif dec ==  5 then decimal = 31      ; 0.3125
   elsif dec ==  6 then decimal = 37      ; 0.375
   elsif dec ==  7 then decimal = 43      ; 0.4375
   elsif dec ==  8 then decimal = 50      ; 0.50
   elsif dec ==  9 then decimal = 56      ; 0.5625
   elsif dec == 10 then decimal = 62      ; 0.6250
   elsif dec == 11 then decimal = 68      ; 0.6875
   elsif dec == 12 then decimal = 75      ; 0.75
   elsif dec == 13 then decimal = 81      ; 0.8125
   elsif dec == 14 then decimal = 97      ; 0.8750
   elsif dec == 15 then decimal =  3      ; 0.9375
   end if               ; aceasta a fost partea zecimală

   if msb == 0 then temperature = units
   elsif msb == 1 then temperature =  16 + units
   elsif msb == 2 then temperature =  32 + units
   elsif msb == 3 then temperature =  48 + units
   elsif msb == 4 then temperature =  64 + units
   elsif msb == 5 then temperature =  80 + units
   elsif msb == 6 then temperature =  96 + units
   elsif msb == 7 then temperature = 112 + units
   end if               ; aceasta a fost partea întreagă

  hd44780_clear
    if sign then hd44780_line1 hd44780 = "-"
                                    ; temperaturi negative
     else hd44780_line1 hd44780= " " ; temperaturi pozitive
    end if
  print_decimal_2(hd44780, temperature, "0"); partea întreagă
    hd44780 = "."
  print_decimal_2(hd44780, decimal, "0"); partea zecimală
  delay_100ms(3)                        ; delay de afişare
end loop
```

Mecanismul este simplu: se verifică valoarea parametrului de intrare şi se alocă acestuia în mod convenabil valoarea reală a temperaturii de ieşire. Pentru simplificare, mecanismul se repetă atât la nivelul părţii zecimale (zecimea + sutimea de °C) cât şi a părţii întregi (°C şi zeci de °C). Metoda simplifică mult calculele matematice care pot fi făcute şi într-o altă manieră mai sofisticată, dar necesită o alegere convenabilă a momentelor comparaţiei (implică detalierea tabelului 7-8 la nivel de °C, pe întregul domeniu măsurat de senzor).

7.2.2 Conversia unui număr zecimal fracţionar în formatul binar cu virgulă fixă

Exemplul următor reprezintă răspunsul dat de un utilizator de microcontrolere mai experimentat, la o întrebare pusă de autor (aflat în postura de începător în microcontrolere). Răspunsul [1] este cea mai bună soluţie la întrebarea: cum se poate reprezenta în binar numărul 3.578 ?

- Se separă partea întreagă de cea zecimală 3d = 11b
- Se înmulţeşte partea zecimală cu 256: 0.578*256 = 147.968
- Partea întreagă a rezultatului se converteşte în binar:
  147 = 128*1+64*0+32*0+16*1+8*0+4*0+2*1+1*1= 10010011b
- Se combină partea întreagă şi cea fracţionară şi se obţine: 3.578 = 11.10010011b

Dacă se doreşte o rezoluţie mai bună (reprezentarea părţii fracţionare pe mai mult de 8 biţi) se continuă conversia restului rămas: 0.968 * 256 = 247.808
Partea întreagă se converteşte din nou în binar:

247:2=123 (rest=1)
123:2=61 (rest=1)
61:2=30 (rest=1)
30:2=15 (rest=0)
15:2=7 (rest=1)
7:2=3 (rest=1)
3:2=1 (rest=1)
1:2=0 (rest=1)

247d = 11110111b (numărul se formează din restul operaţiei de conversie zecimal-binar, ultimul rest rezultat fiind MSB, conform teoriei de conversie zecimal-binar care se învaţă în clasa a V-a)

3.578 = 11.10010011_111101111

Dacă e nevoie de o reprezentare pe 24 de biţi a părţii fracţionare, se continuă raţionamentul pentru 0.808*256=206.848 şi rezultă:

3.578 = 11.10010011_11110111_11001110

Raţionamentul poate fi continuat pentru 32 de biţi sau mai mult.

7.2.3 Conversia complementului față de 2 în binar

Tabelul 7-9 ce reprezintă partea zecimală a temperaturii generate de DS18B20 este corect doar pentru temperaturi pozitive. Pentru a corecta eroarea ce apare în cazul temperaturilor negative, putem realiza o conversie din complement față de 2 în sistem binar:

```
DS1820_start_temperature_conversion ; porneşte conversia
DS1820_read_temperature_raw ( msb, lsb )
 ; citeşte octetul msb şi lsb al temperaturii
procedure comp2_binary
  asm bcf status_c      ; curăță carry
  asm rlf msb, f        ; semnul este în carry
 if status_c then
   sign = high          ; valoare negativă, trebuie convertită
   asm comf msb, f      ; {1} complement,
   asm incf msb, f      ; {2} şi increment = valoare pozitivă
  else sign = low       ; valoare pozitivă, păstreaz-o
 end if
end procedure
```

Conversia propriuzisă este executată în liniile {1} şi {2}, celelate proceduri având rolul de a starta conversia, de a citi valorile *msb* şi *lsb* şi de a testa dacă semnul lui *msb* este 1 (valori de temperaturi negative) sau 0 (temperaturi pozitive).

7.2.4 Conversia binar-ASCII, ASCII-binar

Conversia din binar sau hexagesimal în ASCII este dependentă de numărul de biți ce trebuie convertit. In cap.6.3.1 s-a utilizat în mod repetat conversia din formatul BCD împachetat, în formatul ASCII. Iată cum arată o variantă a conversiei inverse, ASCII-binar:

```
procedure ascii_hex_to_binary (byte in ASCII, byte out bin)is
assembler                ; ASCII poate fi 0,1,...E,F
    bank movf ASCII, w   ; caracterul ASCII este în W
         sublw   "9"     ; scade din valoarea ASCII pe "9”
         movlw   55      ; "A"=65d=41h, 65d-10d(0…9)=55d
         skpnc           ; carry din scăderea anterioară ?
         movlw  "0"      ; nu, 48d=30h în w
    bank subwf ASCII,w   ; da, scade din ASCII valoarea lui W
    bank movwf bin       ; şi mută-l în rezultat
  end assembler
end procedure
```

In exemplul anterior ASCII este reprezentat pe 8 biți (nu este împachetat), rezultatul fiind generat pe *nibble*-ul cel mai puțin semnificativ al octetului bin.

Simplitatea extraordinară pe care o conferă limbajul de programare Jal, este evidențiat în rutina următoare, unde acelaşi algoritm de conversie utilizat anterior apare într-o formă mult mai logică:

```
function ascii_bin (byte in data) return byte is
    if data > "9" then data=data-55 else data=data-48 end if
    if data > 15 then ; aici utilizatorul semnalizează eroare
    end if
    return data
end function
```

respectiv conversia inversă din binar împachetat pe doi ASCII:

```
procedure hex_ascii (byte in data, byte out a1, byte out a2)
    a2 = data & 0x0f  ; mascare cu 0F
    a1 = (data >> 4) & 0x0f ; rotire şi mascare cu 0F
    if a1 > 9 then a1 = a1 + 55 else a1 = a1 + 48 end if
    if a2 > 9 then a2 = a2 + 55 else a2 = a2 + 48 end if
end procedure
```

7.3 Algoritmi matematici

7.3.1 Adunarea şi scăderea numerelor reprezentate pe 16/24 biţi

Una din variantele operării cu doi octeţi, pentru operaţii de adunare şi scădere este cea din exemplul următor. Operanzii a şi b sunt organizaţi pe doi octeţi, ca a_LSB, a_MSB, b_LSB, b_MSB.

```
include 16f84_10
include jpic

-- a = a + b, rezultatul se regăseşte în operandul a
procedure _16b_add  ( byte in b_hi, byte in b_lo,
                      byte in out a_hi, byte in out a_lo ) is
assembler
    MOVF   b_lo,W        ;  b_LSB este în registrul W
    ADDWF  a_lo,f        ;  b_LSB + a_LSB, dacă e depăşire se setează status_C
    MOVF   b_hi,W        ;  b_MSB în W
    BTFSC  status_C      ;  testează depăşirea
    INCFSZ b_hi,W        ;  nu a fost depăşire, b_MSB=b_MSB+1
    ADDWF  a_hi,f        ;  a fost depăşire, adun-o cu a_MSB
end assembler
end procedure

-- a = a - b, rezultatul in a
procedure _16b_sub  ( byte in b_hi, byte in b_lo,
                      byte in out a_hi, byte in out a_lo ) is
assembler
    MOVF   b_lo,W        ;  b_LSB în W
    SUBWF  a_lo,f        ;  a_LSB-b_LSB, status_C la împrumut
```

```
    MOVF   b_hi,W          ; b_MSB în W
    BTFSS  status_C        ; testează dacă a fost împrumut
    INCFSZ b_hi,W          ; nu a fost împrumut, b_MSB=b_MSB+1
    SUBWF  a_hi,f          ; a_MSB-b_MSB
end assembler
end procedure

-- secvență de test a rutinelor
-- 978-496=482
-- 03d2h-01f0h=01e2h
var byte hi = 0x_03
var byte lo = 0x_d2

_16b_sub ( 0x_01, 0x_f0, hi, lo )
pragma test assert hi == 0x_01
pragma test assert lo == 0x_e2
var byte hi1 = 0x_03
var byte lo1 = 0x_d2
-- 03d2h+0178h = 054ah
_16b_add ( 0x_01, 0x_78, hi1, lo1 )
pragma test assert hi1 == 0x_05
pragma test assert lo1 == 0x_4a
pragma test done
end
```

Este evident că se pot modifica aceste rutine pentru adunări şi scăderi pe 24 sau 32 de biți, utilizând un mecanism simplu de multiplicare a regiştrilor necesari, utilizând acelaşi algoritm cu cel prezentat anterior. De exemplu x24, y24, x16, y16, x8, y8 sunt cele şase perechi de octeți participanți la adunare, cel cu indice 24 este cel mai semnificativ iar cel cu indice 8 este cel mai puțin semnificativ, rezultatul respectă şi el această notație:

```
include 16f628_4
include jpic628

var byte x24 = 0x23, x16 = 0xfa, x8 = 0xff
var byte y24 = 0x8c, y16 = 0xcc, y8 = 0x48
var byte rez24, rez16, rez8

procedure add2424(byte in x_24, byte in x_16, byte in x_8,
        byte in y_24, byte in y_16, byte in y_8,
        byte out sum_24, byte out sum_16, byte out sum_8) is

  assembler
     movfw   x_8       ; w <-- x_8
     addwf   y_8, w    ; w <-- x_8 + y_8
     movwf   sum_8     ; w --> sum_8
     movfw   x_16      ; w <-- x_16
     skpnc             ; if carry then
```

```
      incfsz  x_16, f  ; x_16 = x_16 + 1 end if
      addwf   y_16, w  ; w <-- x_16 + y_16
      movwf   sum_16   ; w --> sum_16
      movfw   x_24     ; w <-- x_24
      skpnc            ; if carry then
      incfsz  x_24, f  ; x_16 = x_16 + 1 end if
      addwf   y_24, w  ; w <-- x_24 + y_24
      movwf   sum_24   ; w --> sum_24
   end assembler
end procedure

add2424(x24, x16, x8, y24, y16, y8, rez24, rez16, rez8)
pragma test assert rez24 == 0xb0
pragma test assert rez16 == 0xc7
pragma test assert rez8 == 0x47
pragma test done
```

Rutina de scădere pe 24 de biți, care este asemănătoare cu procedura de adunare se găseşte în biblioteca matematică de pe CD.

7.3.2 Inmulțirea şi împărțirea unui octet cu un număr întreg

Deşi înmulțirea şi împărțirea este soluționată de compilator (pentru rezultate ce pot fi reprezentate pe un octet), principiul de bază utilizat în PIC pentru înmulțire şi împărțire cu un număr întreg este rotirea octetului cu 2^n, urmat de adunarea sau scăderea propriei valori la rezultat. De exemplu fie octetul 0001_0011 = 0x13 = 19d. Să presupunem că dorim înmulțirea lui cu 3. Avem două opțiuni:

1. Rotirea octetului la stânga (înmulțire cu 2^1) şi adunare cu valoarea inițială:
   0001_0011 << 1 = 0010_0110
   0010_0110 +
   0001_0011
   0011_1001 = 0x39 = 57d

Regula adunării binare: 0 + 0 = 0; 0 + 1 = 1; 1 + 1 = 0, transport de rang superior 1

2. Rotire octetului la stânga de două ori (înmulțire cu 2^2) şi scăderea din rezultat a valorii inițiale:
   0001_0011 << 1 = 0010_0110
   0010_0110 << 1 = 0100_1100
   0100_1100 –
   0001_0011
   0011_1001 = 0x39 = 57d

Regula scăderii binare: 0 – 1 = 1, împrumut din rang superior; 1 – 1 = 0; 1 – 0 = 1; 0 – 0 = 0
In cazul împărțirii cu alt număr decât o putere a lui 2, se procedează identic, singura deosebire fiind rotația octetului spre dreapta. Este esențial să ştim că rotirea orcărui registru f, se face prin bitul Carry din registrul STATUS:

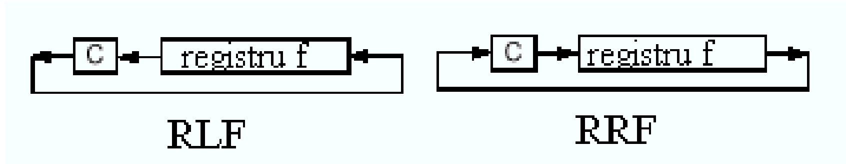


fig.7- 1 Rotirea spre stânga (RLF) şi spre dreapta (RRF) a registrului f

In Jal, pentru a standardiza operația, instrucțiunea >> sau << resetează flagul STATUS_C înaintea rotației propriuzise. Dacă se urmăreşte încărcarea cu o valoare specifică a registrului f (de exemplu transferul LSB în MSB prin rotire, într-un algoritm de 16 biți) operația nu se poate efectua decât în assembler.

7.3.3 Inmulțirea sau împărțirea unui octet cu o constantă fracționară

Din punctul de vedere al algoritmului matematic, înmulțirea şi împărțirea cu un număr fracționar constant este acelaşi lucru. Dacă notăm numărul ca x.yz (constant, valoarea lui rămâne neschimbată indiferent de parametrii de intrare/ieşire în/din PIC), împărțirea cu el este o înmulțire cu 1/x.yz. Desigur că reprezentarea numărului fracționar trebuie să aibă rezoluția maximă pentru a minimiza eroarea de calcul. De aceea formatul în care se converteşte constanta poate fi de exemplu 32Q32. (32 de biți partea întreagă şi 32 de biți partea zecimală). Dacă ne reamintim că orice număr binar N se poate scrie ca o putere a lui 2, atunci :

$$N(32Q.32) = I_{31}*2^{31} + I_{30}*2^{30} + \ldots + I_0*2^0 + F_{-1}*2^{-1} + F_{-2}*2^{-2} + \ldots + F_{-32}*2^{-32}$$

Unde I_k = valoarea bitului k a părții întregi
F_k = valoarea bitului k a părții fracționare

In situația noastră pentru a înmulți un număr variabil cu o constantă, variabila se înmulțeşte cu fiecare element al sumei în care a fost reprezentat numărul constant, excepție fac elementele sumei a căror valoare este 0 şi care pot fi omise. Aşa cum am văzut, înmulțirea cu o putere a lui 2 se traduce printr-o rotație spre stânga, împărțirea cu o rotație spre dreapta, numărul rotirilor este egal cu exponentul puterii lui 2. Revenind la modul de reprezentare al numărului fracționar din cap.7.22:

3.578d = 11.10010011_11110111_1100 ...b

Un alt mod de a reprezenta acest număr este:

3.578 = 2 + 1 + 1/2 + 0 + 0 + 1/16 + 0 + 0 + 1/128 + 1/256...

Dacă dorim să înmulțim un număr variabil v (variabil din procesul pe care microcontrolerul îl supervizează, de exemplu un rezultat al conversiei AD interne) va trebui să executăm cu această variabilă, o sumă de rotații corespunzătoare expresiei anterioare:

v * 3.578 = (v << 1) + v + (v >> 1) + (v >> 4) + (v >> 7) + (v >> 8) + ...

Lungimea părții fracționare depinde de eroarea maximă acceptată pentru reprezentare. Astfel o operație complicată se transformă în simple rotații şi adunări. De real folos utilizatorului de microcontrolere este programul on-line [1], ce transformă inmulțirea cu o constantă în cod asamblor pentru familia PIC.

7.3.4 Inmulțirea numerelor întregi reprezentate pe 8 biți

Inmulțirea este realizată printr-o rotire urmată de o adunare repetată. Compilatorul are predefinit operatorul înmulțirii:

```
operator * ( byte in a, byte in b ) return byte is
      var byte x = 0
      while b > 0 loop
         x = x + a   b = b - 1
      end loop
      return x
end operator
```

Acest operator se comportă ca o funcție, returnând rezultatul înmulțirii dintre a şi b. Atât timp cât b>0 se execută o operație succesivă de adunarea la rezultat a deînmulțitului a şi o decrementare a înmulțitorului b. Atât a,b cât şi rezultatul înmulțirii sunt întregi pe maxim 8 biți. Analiza mecanismului înmulțirii se poate face mai elegant în assembler, unde rezultatul operației poate fi un cuvânt de 16 biți:

```
; dinm   = deînmulțit pe 8 biți
; inm    = înmulțitor pe 8 biți
; H_byte = MSB al rezultatului de 16 biți
; L_byte = LSB al rezultatului de 16 biți
; count  = registru numărător

include 16f84_4
include jpic

procedure mul8 ( byte in dinm, byte in inm,
                 byte out H_byte, byte out L_byte ) is
var byte count = 8
H_byte = 0
L_byte = 0

 assembler
   local loop
        movf    dinm, W       ; deînmultit în w
        bcf     STATUS_C      ; curata status_C
loop:   rrf     inm, F        ; înmultitorul deplasat dreapta
        btfsc   STATUS_C      ; bitul rotit = 0?
```

```
        addwf   H_byte, F       ; nu, aduna cu MSB
        rrf     H_byte, F       ; si deplaseaza dreapta MSB
        rrf     L_byte, F       ; deplaseaza dreapta LSB
        decfsz  count, F        ; s-au terminat 8 biti?
        goto    loop            ; nu, reia
        retlw   0               ; da, gata
 end assembler
end procedure

var byte MSB, LSB
mul8 (0xFF, 0x03, MSB, LSB)
pragma test assert MSB == 2
pragma test assert LSB == 0xFD  ; 0xff * 0x03 = 0x2fd
pragma test done
```

7.3.5 Impărțirea numerelor întregi reprezentate pe 8 biți

Impărțirea numerelor întregi reprezentate pe 8 biți este realizată în Jal prin operatorul împărțirii. Condiția de funcționare este ca împărțitorul (b) să fie nenul iar deîmpărțitul (a) mai mare decât împărțitorul (b). Se execută o scădere repetată a împărțitorului din deîmpărțit, urmat de o incrementare a câtului. Restul nu este disponibil utilizatorului. Simplitatea operației se datorează faptului că rezultatul unei împărțiri a două numere de 8 biți este în mod sigur un număr mult mai mic decât 8 biți.

```
operator / ( byte in a, byte in b ) return byte is
      var byte d = 0
      if b != 0 then
         while a >= b loop
            a = a - b
            d = d + 1
         end loop
      end if
      return d
end operator
```

Bineînțeles că împărțirea se poate efectua şi utilizând elemente de asamblor:

```
include 16f628_4

var byte s0, s1

function div88 ( byte in deîmpărțit, byte in împărțitor, byte
                 out rest) return byte is
```

; deîmpărțit(dividend) = împărțitor(divisor) * cât(quotient) ; + rest(remainder)
; deîmpărțit : împărțitor = cât, rest

```
    var volatile byte cât = 1
    rest = 0
```

```
    status_c = low
   assembler
   local loop
loop:       rlf     deîmpărţit, f
            rlf     rest, f
            movf    împărţitor, w
            subwf   rest, w
            skpnc
            movwf   rest
            rlf     cât, f
            btfss   status_c
            goto    loop
   end assembler
return cât
end function

s1 = div88(35, 10, s0)
pragma test assert s1 == 0x03
pragma test assert s0 == 0x05
pragma test done
```

Funcţia div88 returnează câtul împărţirii a două numere de 8 biţi, pentru simularea anterioară 35d : 10d = 3 rest 5.

7.3.6 Inmulţirea numerelor întregi reprezentate pe 16 biţi

Inmulţirea numerelor este o adunare repetată indiferent de numărul de biţi necesari pentru rezultat. Utilizând convenabil algoritmul de adunare pe 16 sau 24 de biţi descris anterior (sau chiar unul pe 32 de biţi conceput de cititor printr-o simplă modificare a celui de 24 biţi) vom avea disponibilă o rutină elegantă de înmulţire pe 16 biţi:

```
procedure mul16 (byte in x16, byte in x8,
                 byte in y16, byte in y8,
        byte out prod24, byte out prod16, byte out prod8) is
 prod24 = 0 prod16 = 0 prod8 = 0 ; curăţă rezultatul

 for y16 loop ; adunarea repetată a octeţilor semnificativi, cu indici 24 şi 16
  add2424(prod24, prod16, prod8, x16, x8, 0,
          prod24, prod16, prod8)
 end loop

 for y8 loop ; adunarea repetată a octeţilor mai puţin semnificativi cu indici 16 şi 8
  add2424(prod24, prod16, prod8, 0, x16, x8,
          prod24, prod16, prod8)
 end loop
end procedure
```

```
var byte prod24, prod16, prod8
mul16 (0x03, 0x45, 0x03, 0x44, prod24, prod16, prod8)
 pragma test assert prod24 == 0xA
 pragma test assert prod16 == 0xAD
 pragma test assert prod8 == 0x54
 pragma test done
```

Trebuie totuşi ca utilizatorul să verifice depăşirea celor 24 de biți. De exemplu înmulțirea numerelor FFFF * FFFF = FFFE0001 va genera rezultatul pe doar 24 de biți:

```
mul16 (0xff, 0xff, 0xff, 0xff, prod24, prod16, prod8)
 pragma test assert prod24 == 0xfe
 pragma test assert prod16 == 0x00
 pragma test assert prod8 == 0x01
 pragma test done
```

7.3.7 Impărțirea numerelor întregi reprezentate pe 16 biți

Dacă înmulțirea este o adunare repetată, evident că împărțirea va fi o scădere repetată. Cu conditia ca descăzutul să nu fie 0 sau mai mic decât scăzătorul. In biblioteca matematică se găsesc câteva rutine de împărțire realizate în assembler. Desigur că utilizarea unui algoritm de scădere repetată cu o rutină de scădere pe 16 biți este posibilă, deoarece rezultatul împărțirii a două numere de 16 biți va fi în mod cert un număr mai mic de 16 biți (tipic va fi maxim 8 biți).

7.3.8 Compararea a două numere de 16 biți

Dacă este necesară compararea a două numere de 16 biți şi executarea unei anumite ramuri de program în funcție de rezultatul comparării, una din soluții este cea din rutina următoare:

```
-- word hi:lo 16b cuvântul de comparat (comparand)
-- comp hi:lo 16b cuvântul cu care se compară (comparator)
-- dacă WORD = COMP se returnează 0
-- dacă WORD > COMP se returnează 1
-- dacă WORD < COMP se returnează 2

procedure compare_16b
            ( byte in hi_word, byte in lo_word,
              byte in hi_comp, byte in lo_comp )  is
var byte compare_rez
assembler
 local comp_hi, comp_lo, zero, one, two

comp_hi:
    movf    hi_comp,w    ; mută în W MSByte comparator
```

```
    subwf   hi_word,f    ; scade W din MSByte comparand
    btfsc   status_z     ; verifică dacă rezultatul e zero
    goto    comp_lo      ; da, compară LSByte
    btfsc   status_c     ; nu, testează carry
    goto    one
    goto    two
comp_lo:
    movf    lo_comp,w    ; in W este LSByte al comparatorului
    subwf   lo_word,f    ; scade W din MSByte de comparat
    btfsc   status_z     ; verifică dacă rezultatul e zero
    goto    zero
    btfsc   status_c     ; testează rezultatul
    goto    one
    goto    two
zero:
    movlw   0
    movwf  compare_rez  ; WORD = COMP
    return
one:
    movlw   1
    movwf  compare_rez  ; WORD > COMP
    return
two:
    movlw   2
    movwf  compare_rez  ; WORD < COMP
    return

 end assembler
end procedure


; program de test
compare_16b (0x_aa, 0x_ff, 0x_aa, 0x_ff )
pragma test assert compare_rez == 0
compare_16b (0x_00, 0x_ff, 0x_00, 0x_fe )
pragma test assert compare_rez == 1
compare_16b (0x_00, 0x_00, 0x_00, 0x_fe )
pragma test assert compare_rez == 2
pragma test done
end
```

7.3.9 Media aritmetică

Media aritmetică oferă utilizatorului şansa de a obține un rezultat curat, chiar dacă informația de intrare conține zgomot suprapus peste semnalul util. Acest lucru este posibil deoarece zgomotul este de obicei simetric, media lui aritmetică fiind aproximativ nulă.

Succesiunile operațiilor sunt adunarea repetată, urmată de împărțire. Se preferă o mediere cu 2^n pentru a simplifica operația de împărțire:

```
include 16f84_4
include jpic

var byte adres_hi = 0b_0000_0011 ; simulăm un rezultat AD = 1023
var byte adres_lo = 0b_1111_1111
var byte next_adres_hi = 0
var byte next_adres_lo = 0

for 8 loop ; adunarea conversiei anterioare cu conversia curentă
 ; ad_convert (adresult_hi, adresult_lo) ; aici are loc conversia reală,
 next_adres_lo = next_adres_lo + adres_lo
 if status_c then next_adres_hi = next_adres_hi + 1 end if
 next_adres_hi = next_adres_hi + adres_hi
end loop
; next_adres_hi:lo contine suma a 8 achizitii AD, 3FF *8 = 1FF8 in exemplul ales

 pragma test assert next_adres_hi == 0x_1F
 pragma test assert next_adres_lo == 0x_F8

for 3 loop ; împărțire cu 2^3 = 8
 status_c = low
 asm rrf next_adres_hi, f
 asm rrf next_adres_lo, f
end loop

 pragma test assert next_adres_hi == 0x_03
 pragma test assert next_adres_lo == 0x_FF
 pragma test done
```

7.4 In loc de încheiere

Dacă sunteți mai fericit, mai deştept şi mai bogat după ce ați citit această carte, munca noastră nu a fost în zadar. Dacă sunteți aici fără să fi înțeles mai multe aspecte prezentate în carte, reluați pasajele respective cu mai multă atenție. Dacă am greşit ceva iar dvs. ați descoperit greşeala, nu ezitați să-mi comunicați. "Errare humanum est, perseverare diabolicum" (a greşi e omeneşte, a persevera în greşeală este diabolic). Insă numai cine nu munceşte nu greşeşte.

Autorii, la finalizarea ediției a-II-a.

Bibliografie:

1. http://www.piclist.com
2. http://www.jallist-yahoogroups.com
3. Calculatoare electronice, Ioan Dancea, Ed. Didactică şi pedagogică Bucureşti 1975